Procédé et système d'établissement automatique d'un modèle global de simulation d'une architecture

L'invention concerne un procédé et un système d'établissement d'un modèle global de simulation d'une architecture. Plus particulièrement, l'invention concerne un procédé de configuration et un système dit "Configurateur" pour mettre en œuvre le procédé.

Avec l'accroissement de la complexité des systèmes hardware, il faut pouvoir traiter, pour la vérification de systèmes ou circuits intégrés en projet, des configurations rassemblant de plus en plus de modèles écrits en langage de description du matériel, par exemple, de type HDL (tels que VHDL ou Verilog, les plus utilisés) et en langage de haut niveau de type HLL (tels que C ou C++), ces langages décrivant, d'une part, les éléments constitutifs du matériel (hardware) et, d'autre part, les modèles constitutifs de l'environnement de simulation.

Nous allons référer comme ¨Configuration¨ un ensemble de modèles logiciels d'éléments dits ¨Composants¨ constituant un modèle global de simulation.

L'invention est utile dans la vérification de la conception des ASICs par simulation de leur fonctionnement, par exemple, dans un environnement identique à ou très proche de leur utilisation finale, le procédé de configuration permettant le choix de composants et de leurs modèles logiciels parmi une pluralité de modèles disponibles pour la constitution de configurations de simulation.

Dans l'art antérieur les configurations sont rigides et traditionnellement préparées "à la main" à l'aide d'éditeurs de texte ou d'éditeurs graphiques, d'après une liste prédéfinie de configurations envisageables. Chaque modification dans le modèle en langage HDL nécessite des corrections manuelles à répercuter dans l'ensemble des configurations. Cela arrive plusieurs fois au cours du développement d'un ASIC et constitue une source d'erreurs et de difficultés de modification entraînant des retards de réalisation. Les modifications de configuration sont souvent sources d'erreurs

difficiles à trouver, la taille de certaines configurations pouvant atteindre des dizaines de milliers de lignes difficilement manipulables manuellement.

Le temps d'écriture et de mise au point d'une configuration peut être très long, rendant difficile la génération et l'introduction de configurations nouvelles dans l'environnement De ce fait, dans le cas d'un environnement contenant beaucoup d'éléments pour lequel il est difficile de prévoir à l'avance toutes les configurations utiles, on renonce souvent à l'utilisation de certaines variantes de configurations facilitant le déverminage (configurations ciblées simplifiées, par exemple).

Ces difficultés sont accentuées dans les environnements de co-simulation, de plus en plus utilisés, où les modèles proviennent de différentes sources et sont écrits en langages de programmation de haut niveau (HLL) tels que C, C++, ou en langages de descriptions du matériel (HDL), tels que VERILOG ou VHDL. Pour le même composant de la configuration, il existe souvent plusieurs modèles (ex : fonctionnel en langage de programmation de haut niveau, comportemental en HDL, synthétisable HDL, etc...) qu'on aimerait utiliser de manière transparente selon les besoins.

De plus, les modèles à connecter présentent souvent, au niveau des interfaces en vis-à-vis, des différences nécessitant des modules d'adaptation. C'est le cas, par exemple, de circuits avec des interfaces d'entrée-sortie sophistiqués pour lesquels, dans un premier temps, on simule le niveau logique du protocole, le niveau physique étant développé vers la fin du projet. Le choix de modules d'adaptation pour des variantes d'interfaces HDL correspondant aux différentes phases de développement du projet constitue un degré de liberté supplémentaire qui complique d'avantage encore l'écriture des configurations.

Une autre difficulté vient du fait que les modèles mixtes de type HDL (par exemple, VERILOG ou VHDL) / HLL (par exemple, C++) développés séparément, doivent être mis à jour de manière cohérente. Dans le cas d'une gestion non automatisée, c'est une source potentielle d'erreurs.

La présente invention vise à limiter un ou plusieurs de ces inconvénients.

A cet effet, l'invention concerne tout d'abord un procédé d'établissement automatique, au moyen d'un système de traitement de données associé à un programme dit Configurateur, pour constituer un modèle global de simulation d'une architecture comprenant des modèles de circuits intégrés en développement pouvant constituer, à l'aide du Configurateur automatique, une machine ou une partie d'une machine et des modèles d'environnement de simulation, permettant de tester et de vérifier le circuit en développement, d'un fichier de définition de configuration de composants de l'architecture, ces composants constituant des blocs fonctionnels déterminés de description des fonctionnalités de circuits intégrés ou de parties de circuits intégrés, les composants étant choisis par l'utilisateur dans une bibliothèque de différents types de composants et une bibliothèque de composants d'environnements pour constituer le modèle global de l'architecture répondant à la spécification fonctionnelle définie dans le fichier de définition de configuration et conforme à la spécification de l'architecture du modèle global spécifié par un fichier de description de l'architecture, procédé caractérisé par le fait qu'il comporte les étapes suivantes :

- lecture du fichier de description d'architecture du modèle global et mémorisation, dans une table de composants et de règles de connexion, dans une table de règles de cohérence de connexions et dans une table de formatage de fichiers source, des informations relatives à l'ensemble des configurations possibles, chaque composant obtenant un nom identifiant, de façon non équivoque, sa position dans l'architecture, ainsi qu'un type parmi plusieurs types (Composants actifs, Blocs de Monitoring et de Vérification, Blocs intermédiaires, Blocs systèmes et Blocs Globaux),

- instanciation des composants, spécifiés dans le fichier de définition de configuration par l'utilisateur-concepteur au moyen d'une liste de composants présents désignés par leurs nom et type et comportant des paramètres ou faisant appel à des procédures, le fichier de définition de la

configuration comprenant un fichier de sélection des composants et de leur type et des indications supplémentaires optionnelles concernant le type d'interface et le serveur concerné par la configuration à générer par le Configurateur, et mémorisation des informations correspondantes dans une table de connexion des instances,

- connexion topologique des instances et mémorisation des informations correspondantes dans une table de connexion des instances.

- connexion physique des signaux d'interface, au niveau de chaque instance des composants, par application d'expressions régulières, mémorisées dans la table de composants et de règles de connexion, portant sur le nom des signaux constituant une table de câblage,

- utilisation de la table de connexion des instances, de la table de câblage et de la table de formatage pour générer automatiquement des fichiers source de type HDL et de type HLL du modèle global de simulation, correspondant à la configuration spécifiée par le fichier de définition de configuration.

Selon une autre particularité du procédé, le système Configurateur transmet aux parties de type HLL de chaque composant des informations sur :

- le nom du composant,
- le type de l'instance,
- le chemin HDL, à savoir le nom hiérarchique du composant dans la description du modèle.

Selon une autre particularité du procédé, le fichier de définition de configuration comporte en plus un mot–clé indiquant le nom ou numéro du serveur sur lequel se trouve instancié un composant dans le cas d'une utilisation du procédé sur un système multi-serveur.

Selon une autre particularité du procédé, dans le cas d'une utilisation multi-serveur, le système Configurateur exécute les étapes suivantes :

- découpage de la Configuration en plusieurs parties (de type HDL et de type HLL) en triant les composants de type HDL

et les objets de type HLL selon leurs appartenances aux serveurs,

- génération des composants de type HDL périphériques servant à l'envoi et la réception des signaux entre les parties de la configuration,
- duplication des Blocs Globaux par le système Configurateur et instanciation des Blocs Globaux dupliqués sur chaque serveur,
- génération des parties de type HLL servant de support de communication entre les serveurs.

Selon une autre particularité du procédé, la connexion automatique entre les composants par le système Configurateur comprend plusieurs phases :

- une phase topologique de haut niveau réalisant la sélection des composants et leurs positionnements respectifs,
- une phase de câblage réalisant la connexion effective entre les composants, cette phase générant comme résultat une table de câblage associant les signaux connectés ensemble, au nom unique du fil les connectant,
- une phase de génération des fichiers sources de type HDL et de type HLL.

Selon une autre particularité du procédé, la phase de câblage est effectuée par le système Configurateur selon les trois étapes suivantes :

a. les Blocs Globaux et les Blocs Système sont connectés en premier à tous les composants,
b. viennent ensuite les connexions des signaux entre les autres composants,
c. à la fin du câblage, une passe supplémentaire permet de connecter les signaux restant non connectés de chaque composant à des signaux prédéterminés pour assurer un état stable et déterminé, le

système Configurateur générant alors des configurations partielles comprenant un sous-ensemble de l'architecture.

Selon une autre particularité du procédé, les signaux prédéterminés sont les signaux du Bloc Système correspondant au composant.

Selon une autre particularité, le fichier de description de l'architecture du modèle global comprend les modèles de simulation des Blocs globaux et des Blocs Système, ces deux types de composants étant connectés entre eux et gérant des signaux d'environnement.

Selon une autre particularité, les Blocs Système sont connectés aux autres composants et leur fournissent des signaux système qui leur sont spécifiques.

Selon une autre particularité du procédé, le système de traitement de données effectue un contrôle de conformité des connexions en comparant la table de connexion des instances réelles entre blocs avec la table de règles de cohérence de connexions.

Selon une autre particularité du procédé, le système de traitement de données compare les connexions physiques entre les composants, à la table de règles de cohérence de connexions, pour détecter des incompatibilités entre extrémités de connexions entre les composants, et, en pareil cas, il spécifie, et ajoute, dans la table de connexion des instances, un composant adaptateur inséré dans la connexion considérée.

Selon une autre particularité, le fichier de définition de configuration comprend des informations, spécifiées par un attribut, concernant l'utilisation de composants adaptateurs avec les instances des Composants actifs, dont les connexions sont comparées à la table de connexion des instances, pour détecter des incompatibilités entre extrémités de connexions entre les composants, et, en pareil cas, il spécifie, et ajoute, dans la table de connexion des instances, un autre composant adaptateur inséré dans la connexion considérée.

Selon une autre particularité du procédé, le système de traitement de données sélectionne certaines des connexions entre composants de la table de règles de cohérence de connexions et spécifie, et ajoute, dans la table de

connexion des instances, des connexions supplémentaires constituant des dérivations aboutissant à des modèles supplémentaires respectifs, représentant des outils de surveillance des connexions.

Selon une autre particularité du procédé, dans la phase de génération des fichiers sources, le système Configurateur génère les fichiers source en langage HDL et en langage HLL en s'appuyant sur le contenu de la table de composants et de règles de connexion, la table de règles de cohérence de connexions, la table de formatage de fichiers source, la table de connexion des instances et la table de câblage.

Selon une autre particularité du procédé, le système de traitement de données effectue un traitement par le système Configurateur, pour chaque variante de configuration, pour obtenir plusieurs modèles de simulation correspondant à la même spécification fonctionnelle, mais écrits en une description comportant différents mélanges des langages de niveaux différents (HDL, HLL).

Selon une autre particularité du procédé, le système de traitement de données établit la spécification fonctionnelle du modèle global de simulation dans un format informatique compatible avec un langage de programmation de haut niveau (HLL) et en format compatible avec un langage de description du matériel (HDL).

Selon une autre particularité du procédé, le fichier de définition de configuration comprend, pour chaque composant, au moins une partie en langage de type HDL, ladite partie en langage de type HDL fournissant une interface vers d'autres modèles.

Selon une autre particularité du procédé, les modèles comprenant une partie en langage de type HLL comprennent des adaptateurs d'interface.

Selon une autre particularité, le système Configurateur choisit chaque modèle d'adaptateurs d'interface en fonction de la table de règles de cohérence de connexions.

Selon une autre particularité du procédé, les connexions des signaux physiques sont spécifiés par des "Ports", chaque port étant une sélection arbitraire des signaux de l'interface de type HDL d'un composant à l'aide des

expressions régulières portant sur les noms de ces signaux, et étant constitué de paires expression régulière / expression de substitution, ces expression étant appliquées successivement au nom de chaque signal de l'interface de type HDL, et, si la substitution finale est identique pour deux signaux, ces derniers sont connectés entre eux, la connexion étant mémorisée dans la table de câblage.

Selon une autre particularité du procédé, chaque adaptateur d'interface étant partagé entre plusieurs modèles connectés sur le même port, un seul de ces modèles émet des signaux sur ledit port.

Un autre but de l'invention est de proposer un système Configurateur qui permette de pallier un ou plusieurs des inconvénients précédents.

Ce but est atteint par le système de traitement de données pour l'établissement automatique d'un modèle global de simulation d'une configuration de blocs fonctionnels déterminés, mutuellement reliés par des connexions d'interfonctionnement, pour constituer le modèle global de simulation d'une architecture comprenant des modèles de circuits intégrés en développement pouvant constituer une machine répondant à la spécification fonctionnelle d'une configuration, système caractérisé par le fait que le système de traitement de données utilise un programme Configurateur qui comprend des moyens de réaliser une simulation du câblage par l'application d'expressions régulières mémorisées, d'utiliser un fichier de définition de configuration en langage de haut niveau, une table de composants et de règles de connexion décrivant les propriétés (type, Interfaces de type HDL, ports, constructeurs d'objets de classes HLL, etc...) des composants logiciels de simulation du circuit, une table de règles de cohérence de connexions en langage de haut niveau (HLL), des moyens d'instancier les éléments résultants du fichier de définition de configuration, un générateur du code HLL qui combine les paramètres des composants avec les règles de connexion.

Selon une autre particularité du système, il existe au moins cinq types de composants : les Composants actifs, les Blocs de Monitoring et de

Vérification, les Blocs intermédiaires, les Blocs Système et les Blocs Globaux.

Selon une autre particularité du système, il est agencé pour effectuer un contrôle de conformité des connexions en comparant la table de connexion des instances avec une table de règles de cohérence des connexions physiques entre les modèles choisis des blocs constituant le modèle global.

Selon une autre particularité du système, il est agencé pour comparer la table de connexion des instances réelles entre blocs à la table de règles de cohérence des connexions, pour détecter des incompatibilités entre extrémités de connexions entre blocs, et, en pareil cas, pour spécifier, et ajouter, dans la table de règles de cohérence des connexions réelles, un bloc fonctionnel adaptateur inséré dans la connexion considérée.

Selon une autre particularité du système, la table de composants et de règles de connexion, comprenant les propriétés des composants, contient des paramètres globaux communs à tous les types de composants et se présente sous forme d'une table répartie suivant une ou plusieurs tables, associatives ou non, dont les entrées sont des noms désignant l'ensemble des modèles possibles pour un même composant.

Selon une autre particularité du système, les tables associatives peuvent contenir la description, soit sous forme de suites de paramètres, ou bien sous forme de références à des procédures qui génèrent les valeurs requises, les entrées de ces tables associatives étant des noms désignant l'ensemble des modèles possibles pour un même composant et formant une chaîne de caractères contenant des identificateurs spéciaux prédéterminés, substitués par les valeurs calculées par le système Configurateur.

Selon une autre particularité du système, au moins trois sélecteurs indiquent l'instance à prendre en compte, cette dernière étant transmise comme paramètre à un constructeur d'un objet HLL :

- un premier sélecteur indique l'instance courante,
- un deuxième sélecteur précise l'instance connectée à l'autre extrémité du port,

- un troisième sélecteur indique l'instance composée correspondant au Composant actif contenant le port d'observation.

Selon une autre particularité du système, le système Configurateur utilise une ou plusieurs tables de règles de cohérence de connexions, qui représentent les règles d'interconnexion entre les composants et d'insertion des composants intermédiaires, une ou plusieurs tables de composants et de règles de connexions, qui représentent les règles de connexion niveau système et les règles de génération de connexions entre les signaux, et une ou plusieurs tables de formatage de fichiers source, qui représentent les règles de génération des instances des objets de type HLL.

Selon une autre particularité du système, le système Configurateur utilise :

- une classe de base en HLL permettant d'identifier de façon univoque chaque objet instancié et d'interroger la configuration,

- des moyens de génération et d'instanciation automatique des Blocs Système,

- des moyens des tableaux associant les signaux connectés ensemble au nom unique du fils connectant,

- des moyens d'utiliser un tableau de formatage pour générer les fichiers sources en HDL et HLL.

Selon une autre particularité du système, l'opérateur spécifie fonctionnellement, dans la plus large mesure possible, la configuration dans le langage de plus haut niveau et il complète la spécification fonctionnelle par les composants en langage de plus bas niveau.

Selon une autre particularité du système, les entrées suivantes du hachage définissent le Type du composant (par exemple DUT (modèle HDL), XACTOR (transactor), MONITOR, VERIFIER ou autres) et, à chaque Type de Composant, correspond un hachage composé à son tour des entrées suivantes :

- une première entrée *ReqModule* - contient le nom du module HDL du composant et le nom du fichier source correspondant,

- une deuxième entrée *Connect* - est la définition de la méthode de sélection des signaux faisant partie d'un Port, cette description étant composée d'une suite d'entrées indexées par le nom du Port ; à chaque nom de Port, le configurateur associe un tableau d'expressions régulières et un pointeur sur une procédure de connexion des signaux qui gère l'application de ces expressions aux noms des signaux de l'interface du composant.

Selon une autre particularité du système, la structure générique des Composants actifs comprend un Bloc contenant la description HDL et un Bloc en HLL, fournissant les chemins d'accès aux ressources HDL et, le cas échéant, une description du bloc en HLL, l'ensemble des signaux du bloc HDL constituant l'interface du Bloc englobant, formée, d'une part, de Ports, qui sont des sélections logiques arbitraires des signaux d'une interface et, d'autre part, d'adaptateurs d'interface, qui sont les parties logicielles assurant, sur chaque Port, la communication bi-directionnelle entre les parties en HLL et celles en HDL, les adaptateurs d'interface étant choisis par le système Configurateur.

Selon une autre particularité du système, les Ports sont spécifiés sous forme d'expressions régulières, qui servent à la fois à sélectionner les sous-ensembles de signaux à connecter et à définir les règles de connexions.

Selon une autre particularité du système, le système Configurateur génère des Composants dits de Transfert qui sont insérés de chaque côté de la coupure sur les serveurs, ces composants étant simplement des fils pour les entrées et des registres pour les sorties.

Les composants à modèle élémentaire qui sont partagés entre les Composants à Modèle Composé ou les Blocs Globaux appartenant aux différents serveurs sont automatiquement dupliqués par le système Configurateur et instanciés sur chaque serveur.

L'invention sera mieux comprise à l'aide de la description suivante d'un mode préféré de mise en œuvre du procédé de l'invention, en référence aux dessins annexés, sur lesquels :

- la figure 1 représente, sous forme très schématique, un exemple d'architecture du modèle global de simulation pour un circuit intégré (ASIC) en développement appelé BRIDGE (12) ;
- les figures 2a à 2d sont des diagrammes illustrant les différentes composantes du système Configurateur et les étapes de mise en œuvre de ces composantes dans le procédé de l'invention ;
- les figures 3a à 3c représentent différents stades de modélisation d'un circuit à l'aide d'un modèle mixte de type HDL (VERILOG ou VHDL) et de type HLL (C++) ;
- la figure 4 représente la structure générique d'un Modèle élémentaire ;
- la figure 5 représente la structure générique d'un Modèle Composé ;
- la figure 6 décrit la correspondance entre les tables de la description du système configurateur sur les figures 2a à 2d et les tables de la mise en œuvre du procédé de l'invention ;
- les figures 7a à 7k représentent schématiquement les différents modèles des composants avec leurs variantes d'interfaces et de niveau de description nécessaires pour les configurations relatives à l'architecture de l'exemple de la figure 1 ;
- les figures 8a à 8e représentent les configurations successives de l'architecture aboutissant à une constitution du modèle donnée dont le modèle final correspond à celui de la figure 8e et pour lequel tous les modèles de type HDL du circuit en projet sont disponibles, ce modèle final correspondant à la configuration.
- la figure 8f représente la répartition de la configuration de simulation du modèle de la figure 8a, par exemple sur deux machines.

L'invention concerne un système dit "Configurateur" et le procédé de configuration mis en œuvre par le système.

Un modèle global de simulation est typiquement composé d'un ou plusieurs modèles de circuits intégrés testés (DUT) entourés de modèles qui créent un environnement de test et vérification. Ces modèles créent des stimuli complexes et reçoivent des réponses complexes du modèle testé. Ces composants peuvent être des transactors (XACTORS) – des modèles possédant généralement une interface programmatique (API) permettant un pilotage par des tests externes au modèle, ces tests étant écrits généralement en langage de haut niveau (HLL).

L'environnement de vérification peut contenir aussi des composants dits Bloc de Monitoring (MONITOR) et des composants dits Bloc de Vérification (VERIFIER). Ces composants n'interviennent pas directement dans l'échange de signaux entre les autres constituants du modèle global de simulation mais servent à les observer et à interpréter. Les Blocs de Monitoring (MONITOR) servent de support d'analyse pour les tests, ils possèdent des interfaces programmatiques (API) pour signaler des événements observés sur les signaux de modèle global. Les Blocs de Vérification (VERIFIER) sont des composants qui possèdent une spécification de référence de fonctionnement de modèle testé et, en observant les signaux du modèle global de simulation, sont capables de vérifier le bon fonctionnement du modèle.

La figure 1 représente un exemple d'architecture d'un système de circuit intégré en développement dont on veut, dans une première étape, mettre au point le modèle global de simulation correspondant à la figure 8c, choisie pour illustrer le nombre le plus important de mécanismes mis en œuvre par le configurateur. Il doit être clair que les étapes des figures 8a, 8b pourraient être d'abord exécutées selon la disponibilité des modèles et les objectifs de vérification. Le modèle final souhaité est celui correspondant à la figure 8e. L'architecture est constituée d'un processeur (CPU) communicant par une passerelle (BRIDGE) avec une mémoire système (MEMORY) et des entrées sorties (I/O). Le modèle auquel aboutit le système "Configurateur" de l'invention est constitué d'un composant processeur CPU de type XACTOR

relié par une interface de type "fbus_p" à un bloc intermédiaire (fbus_xchg) 101 ayant une interface de type différent. Un autre bloc intermédiaire (fbus_xchg) 102 figures 8b et 8c relie le premier bloc intermédiaire 101 à un composant de type passerelle (BRIDGE) de type DUT_CORE qui communique, d'une part avec un modèle majoritairement en langage de type HDL d'une mémoire (MEMORY) de type DUT, d'autre part avec un modèle majoritairement en langage de type HDL d'une entrée/sortie (I/O) de type DUT et enfin avec un bloc système (SYS_BRIDGE).

Le système "Configurateur" de l'invention gère la connexion d'éléments logiciels de simulation appelés composants pour les besoins de la simulation des circuits hardware.

Il existe au moins 5 classes de composants :

- Les "Composants actifs" (voir ci-après) ;
- Les "Blocs de Monitoring et de Vérification" (voir ci-après) ;
- Les "Blocs intermédiaires" (voir ci-après) ;
- Les "Blocs Système" (voir ci-après) ;
- Les "Blocs Globaux" (voir ci-après et 1 de A2).

Chaque type de composant peut posséder plusieurs variantes de réalisation d'un même élément, différenciées par le niveau de description (voir ci-après) ou par les signaux sur les interfaces (voir ci-après). Chaque type de Composant peut être décrit à plusieurs niveaux de détails (fonctionnel, comportemental, portes, etc...) en langage de type HDL, tel que VERILOG ou VHDL, ou en langage de haut niveau (HLL), tel que C ou C++ complété d'une interface de type HDL.

Plusieurs niveaux de descriptions peuvent coexister pour décrire des fonctionnalités similaires et avoir des interfaces de type HDL similaires mais pas forcément identiques. Certaines descriptions peuvent avoir plus de fonctionnalités, et les interfaces de type HDL peuvent contenir des jeux de signaux complètement différents.

Les composants sont connectés selon un schéma prédéterminé considéré fixe pour chaque exécution du système Configurateur. Ces connexions sont définies par l'architecture du modèle global de simulation et spécifiées par le fichier de description d'architecture (FDARCH) (voir annexes A1-A5).

Chaque instance d'un composant dans ce schéma obtient un nom ou label qui identifie de façon non équivoque la position du composant et son type.

Le fichier de définition de la configuration (FCONF) fournit une description synthétique de la variante de Configuration à générer par le système Configurateur. Seuls les composants principaux (Composants Actifs, Blocs de Monitoring et Blocs de Vérification) y sont mentionnés avec les types de modèles souhaités. Les autres composants (Blocs Globaux, Blocs Système et Blocs Intermédiaires) seront instanciés automatiquement par le système configurateur.

Parmi les différents types de composants, les "Composants Actifs" constituent le sous-ensemble des objets explicitement désignés dans le fichier de configuration (FCONF) et qui échangent des stimuli en émission et réception avec leur environnement.

Parmi les différents types de composants, les "Blocs de Monitoring et Vérification" constituent le sous-ensemble des objets, explicitement désignés dans le fichier de configuration (FCONF), qui ne font que recevoir des informations de leur environnement. Ils sont utilisés à des fins d'observation et de Vérification (MONITOR, VERIFIER). La granularité de traitement de ces modèles est l'événement qui rend compte des changements de valeurs des signaux de contrôle et des échanges arbitraires de données.

Tous les autres Composants constituent des composants dits implicites, qui sont gérés et instanciés de façon automatique par le système configurateur, en fonction des paramètres du fichier de configuration (FCONF) (type de composants explicites, type d'interface, ...).

Les composants peuvent être connectés entre eux directement ou par l'intermédiaire de composants d'adaptation externe dits "Blocs

Intermédiaires" définis et déclarés spécialement à cet effet. Ils sont souvent utilisés, comme nous le verrons par la suite, lors des phases successives de développement pour compléter les parties manquantes du design.

Le système Configurateur peut insérer ainsi un ou plusieurs blocs intermédiaires pour connecter deux composants actifs.

Les composants dits "Blocs Systèmes" sont associés aux autres composants pour leur fournir les signaux d'environnement qui leur sont spécifiques. Ces composants encapsulent, au niveau de chaque interface, l'ensemble des signaux système ne participant pas à la connexion entre les autres composants. Les "Blocs Systèmes" sont eux-mêmes connectés aux "Blocs Globaux" et gèrent l'ensemble des signaux d'environnement : les signaux d'horloge et de contrôle général (clock, reset, powergood, diagnostic), qui peuvent être éventuellement transformés pour les adapter au besoins du bloc actif correspondant (changement de polarité, retard etc.), ainsi que les signaux spécifiques, différents pour chaque instance particulière du composant actif concerné, par exemple les signaux codant le numéro de module ou le mode de fonctionnement du composant, etc... Ces derniers sont gérés par des paramètres fournis par le système Configurateur aux instances de modèles des composants générés. Si, pour un Composant donné, le Bloc Système n'est pas nécessaire (ce qui est indiqué par les tables de description de la configuration), il sera relié directement aux Blocs Globaux (cf. 1 de A2).

Le fichier de configuration (FCONF) peut contenir des informations supplémentaires spécifiant des blocs intermédiaires à utiliser en association avec les instances des composants actifs. Ainsi, l'utilisateur peut influencer le choix du composant intermédiaire, ou lever l'ambiguïté, dans le cas de plusieurs choix possibles. Les connexions ainsi obtenues sont comparées à la table de règles de cohérence de connexions (TRCOH) pour détecter des incompatibilités entre extrémités de connexions entre les composants et, en pareil cas, choisir et ajouter, dans la table de connexion des instances (TCINST), encore un composant adaptateur (Bloc Intermédiaire) inséré dans la connexion considérée.

Le système Configurateur s'appuie sur une représentation générique, telle que décrite sur les figures 3a à 3c, commune à l'ensemble des composants déclarés dans le fichier de description d'architecture (FDARCH) du modèle global de simulation, et comprenant deux parties, de type HDL (par exemple VERILOG ou VHDL) et de type HLL (par exemple C++).

Dans le cas d'un composant décrit complètement en langage de type HDL (figure 3a), la partie de type HLL est réduite à une instance qui permet de signaler sa présence dans la Configuration au cours de la simulation et qui fournit les chemins d'accès aux ressources de type HDL du composant.

Pour les composants décrits en langage de type HLL (figure 3c), c'est la partie de type HDL qui est réduite au strict minimum, et qui est limitée à la description des signaux et registres d'interface.

Tous les niveaux intermédiaires entre ces deux extrémités sont possibles et naturellement exploités dans le contexte de processus de développement des circuits ASIC.

Typiquement, au cours d'un développement, on commence par un modèle de type HLL avec une interface de type HDL minimale et, ensuite, on passe à des modèles de type HDL de plus en plus complets écrits par les concepteurs et on termine par les modèles du niveau portes logiques synthétisés automatiquement à partir de modèles de type HDL.

Les modèles mixtes sont souvent utilisés parce qu'ils permettent une simulation précise et efficace en dédiant le langage de haut niveau (HLL) aux modélisations complexes pour lesquelles le langage de type HLL offre une description compacte et rapide à l'exécution. Néanmoins, même pour les modèles écrits majoritairement en langage de type HLL, la partie de type HDL peut être non triviale pour des raisons de pertes de performances dues aux changements de contexte de simulation entre, respectivement, les parties de type HDL et de type HLL. Un exemple typique est une interface dont les signaux changent, même sans activité réelle de transfert de données. Dans ce cas, il est préférable de traiter les signaux dans la partie

de type HDL du modèle, et de passer à la partie de type HLL quand les données sont présentes sur l'interface.

L'interface d'un composant est l'ensemble de tous les signaux présents à sa périphérie. Pour les composants décrits uniquement en langage de type HDL, cela correspond à l'interface externe de la définition de ce composant en langage de type HDL (par ex. VERILOG ou VHDL). Pour les composants définis de manière mixte en langage de types HLL et HDL, l'interface d'un composant est la somme des signaux de tous les modules de type HDL utilisés à la périphérie de ce composant.

La figure 4 décrit la structure générique des modèles élémentaires constituant la majorité des composants. Cette structure s'applique typiquement, mais pas uniquement, au cas du circuit ASIC en développement, des adaptateurs d'interface, blocs intermédiaires, blocs système.

Elle comprend un Bloc englobant noté C contenant la description de type HDL notée A et le Bloc HLL noté B fournissant les chemins d'accès aux ressources de type HDL et, le cas échéant, une description du bloc en langage de type HLL. L'ensemble des signaux du bloc de type HDL constitue l'Interface du bloc C. Pour les besoins de connexion entre les blocs nous utilisons la notion de Ports (voir plus loin) qui sont des sélections logiques arbitraires des signaux d'une Interface. Il est possible qu'un signal appartienne à plusieurs Ports à la fois.

Le système Configurateur est capable de traiter des modèles dits 'composés' comprenant une partie en langage de type HLL et incluant d'autres modèles des composants dits 'adaptateurs d'interface'. Les adaptateurs d'interface sont des modèles mixtes de types HDL / HLL possédant une interface programmatique (API) en langage de type HLL par laquelle ils peuvent communiquer avec le modèle composé. Les modèles composés sont particulièrement utiles pour les composants d'environnement de simulation (par exemple MONITOR, VERIFIER, XACTORS) où le modèle doit s'adapter à des signaux présents sur les différentes variantes des modèles utilisés dans une configuration.

La figure 5 décrit la structure générique de modèles composés, utilisés surtout pour modéliser les constituants de l'environnement de simulation. Vu de sa présentation extérieure, le modèle composé est identique au modèle élémentaire. A l'intérieur, la spécification HLL du modèle (notée D) communique avec l'interface extérieure de type HDL à travers les adaptateurs d'interface (notés $C\_Port)_i$ modélisés au moyen de structures élémentaires en langage de type HLL (notées B_PORT) et en langage de type HDL, (notées A_PORT).

Le système Configurateur est agencé pour sélectionner automatiquement les adaptateurs d'interface pour un modèle composé. Le système Configurateur examine la liste d'adaptateurs d'interface utilisables pour un modèle composé donné décrit par les propriétés du modèle, et choisit le modèle d'adaptateur d'interface dont les connexions physiques avec le reste du modèle global sont en conformité avec la table des règles de cohérence de connexions (TRCOH). Cette approche permet une rapide adaptation à de nouveaux types d'interfaces en ajoutant de nouveaux adaptateurs.

Il est important de souligner qu'un même composant peut aussi bien être modélisé par une structure élémentaire, ou composée, suivant son niveau de description.

Un composant adaptateur d'interface peut être partagé entre plusieurs modèles composés connectés sur le même port. Un seul d'entre ces modèles est autorisé à émettre des signaux sur le port, les autres modèles étant purement observateurs. Une utilisation typique concerne les blocs de Monitoring et Vérification qui n'envoient pas de signaux en sortie. Le partage des adaptateurs d'interface accélère la simulation par la simplification et la factorisation des parties du modèle.

On appellera par la suite Sondes, les adaptateurs d'interface servant à observer les signaux pour le compte des Blocs de Monitoring ou de Vérification qui sont des Modèles Composés.

Le système Configurateur est conçu pour optimiser l'utilisation des Sondes en minimisant leur nombre. Comme la description des Composants

établit la notion d'équivalence entre certains Composants, le système Configurateur essaye d'abord de partager le port d'un "Composant actif". Si cela s'avère impossible, il instancie un composant Sonde adéquat connectable sur l'interface de type HDL à partir d'une liste fournie dans la description du Composant Monitoring ou Vérification.

La partie ci-après décrit les définitions relatives aux connexions entre Composants. Les notions d'interface et de port servent de support à la description des connexions entre les composants.

Nous rappelons que le port est une sélection arbitraire des signaux de l'interface de type HDL d'un composant, réalisée à l'aide des expressions régulières portant sur les noms de ces signaux. Un signal donné peut appartenir à un ou plusieurs ports. La définition de chaque port est constituée de paires de type expression régulière et expression de substitution correspondante. Ces expressions sont appliquées successivement au nom de chaque signal de l'interface et, en cas d'adéquation 'match', l'expression de substitution est appliquée et le nom ainsi obtenu passe au traitement de la substitution suivante. Si la substitution finale donne un résultat final identique pour deux signaux, ces derniers seront connectés ensemble. Le configurateur génère un nom unique pour la connexion et place les informations adéquates dans la table de câblage (TCAB). La méthode décrite permet d'exprimer des règles complexes de connexion entre deux composants. Par exemple on peut connecter des signaux de noms différents, ou créer des règles de connexion des signaux d'entrée avec les signaux de sortie.

La création de cette méthode de définition par les interfaces et les ports permet un découplage entre les déclarations de modules de type HDL et les spécifications de connexions pour le système Configurateur. Ce dernier combine ces deux sources d'information pour générer les connexions. Dans la majorité des cas, les modifications de déclarations dans les parties de type HDL, très fréquentes pendant le développement,

n'entraînent pas de modifications dans les expressions régulières décrivant les câblages.

Dans l'exemple de mise en œuvre de l'invention traité ci-après, seules les connexions point à point sont traitées. Les connexions de type "bus" sont facilement modélisables en utilisant un composant à plusieurs sorties englobant le bus.

Nous allons décrire ci-après le procédé de connexion automatique entre les "composants" par le système Configurateur pour constituer le modèle global de simulation. Ce procédé comprend les étapes suivantes :

- Lecture du fichier de description d'architecture (FDARCH) du modèle global et mémorisation, dans une table de composants et de règles de connexion, dans une table de règles de cohérence de connexions et dans une table de formatage de fichiers source, des informations relatives à l'ensemble des configurations possibles, chaque composant obtenant un nom identifiant, de façon non équivoque, sa position dans l'architecture, ainsi qu'un type parmi plusieurs types (Composants actifs, Blocs de Monitoring et de Vérification, Blocs intermédiaires, Blocs systèmes et Blocs Globaux).
- Instanciation des composants, spécifiés dans le fichier de définition de configuration par l'utilisateur-concepteur au moyen d'une liste de composants présents désignés par leurs nom et type et comportant des paramètres ou faisant appel à des procédures, le fichier de définition de la configuration comprenant un fichier de sélection des composants et de leur type et des indications supplémentaires optionnelles concernant le type d'interface et le serveur concerné par la configuration à générer par le Configurateur, et mémorisation des informations correspondantes dans une table de connexion des instances.
- Connexion topologique des instances selon le schéma défini par la table de composants et de règles de connexions (TCRC).

- Lecture et mémorisation des sources de type HDL des modèles instanciés pour en extraire les noms et types des signaux d'interfaces.
- Câblage des signaux d'interface au niveau de chaque instance des composants par application d'expressions régulières, stockées dans la table de composants et de règles de connexions (TCRC), et portant sur le nom des signaux constituant la table de câblage (TCAB).
- Utilisation de la table de connexion des instances (TCINST), de la table de câblage (TCAB) et de la table de formatage (TFMT) pour générer automatiquement les fichiers source de type HDL (MGHDL) et de type HLL (MGHLL) du modèle global de simulation correspondant à la configuration spécifiée par le fichier de configuration (FCONF).

Le déroulement de la phase topologique est le suivant :

- Les Composants explicites spécifiés dans le fichier de configuration (FCONF) sont instanciés en premier par le système Configurateur et placés dans la table de connexion des instances (TCINST). Cette table contient le descriptif de chaque instance (label, type), accompagné de la liste des composants avec lesquels il est connecté.
- Ensuite, les Blocs Intermédiaires, spécifiés explicitement sur les ports des "Composants actifs" dans le fichier de configuration, sont instanciés et ajoutés dans la table de connexion des instances (TCINST).
- Les connexions entre les composants actifs instanciés sont comparées avec les règles contenues dans la table de cohérence de connexions (TRCOH) pour détecter des incompatibilités entre les extrémités des connexions entre les composants et, dans ce dernier cas, pour choisir et ajouter, dans la table de connexion des

instances (TCINST), un composant adaptateur (Bloc Intermédiaire) à insérer dans la connexion considérée.

- Ensuite sont instanciés les composants de type Bloc de Monitoring et Bloc de Vérification. Le système Configurateur analyse les connexions des composants à modèle composé et cherche des adaptateurs d'interface partageables au niveau des ports commun (voir modèles C_Port_i sur la Figure 3). S'il n'existe pas de composant approprié pour partager la connexion en observation des signaux requis, un composant adaptateur d'interface aux propriétés spécifiées par la description sera instancié par le configurateur. La table de connexion des instances (TCINST) est remise à jour.
- Enfin, les composants 'bloc système' sont instanciés et placés dans la table de connexion des instances (TCINST), pour les composants qui en ont besoin.

En phase de Câblage, le système Configurateur connecte les signaux d'interface au niveau de chaque port par application d'expressions régulières portant sur les noms des signaux comme décrit dans la définition du port. Ces expressions sont appliquées aux noms des signaux extraits à partir des descriptions de type HDL des composants, de telle sorte que les changements des interfaces de type HDL d'une version de modèle à l'autre n'influent pas directement sur les spécifications des connexions entre les blocs. Cette phase génère comme résultat la table de câblage (TCAB) associant les signaux connectés ensemble, au nom unique du fil les connectant. Un certain nombre de vérifications sont alors possibles, portant par exemple sur les connexions sans source, ou la connexion de plusieurs sorties ou autres.

Les différentes étapes de la phase de câblage sont les suivantes :

- Les Composants Globaux et Blocs Système sont câblés en premier à tous les composants.
- Ensuite sont câblés les signaux entre les autres composants.
- A la fin du câblage, une passe supplémentaire permet de connecter les signaux d'un composant non connectés dans les phases précédentes de câblage à des signaux du bloc système correspondants, spécialement prévus pour forcer des valeurs connues et stables inhibant les ports non utilisés. Les signaux concernés des blocs système portent un marquage spécial (par exemple un préfixe spécial dans le nom) pour ne pas être connectés au cours des phases de câblage précédentes.

En phase de génération des sources, le système Configurateur génère les fichiers source de type HDL et de type HLL en s'appuyant sur le contenu de la table de connexion des instances (TCINST), de la table de câblage (TCAB) et de la table de formatage (TFMT) pour générer automatiquement les fichiers source de type HDL (MGHDL) et de type HLL (MGHLL) du modèle global de simulation correspondant à la configuration spécifiée par le fichier de configuration (FCONF).

Le fichier source de type HDL contient une description de la partie de type HDL (par exemple en VERILOG) du modèle global de simulation correspondant au fichier de configuration (FCONF). En particulier, il contient l'instanciation de tous les modèles de type HDL des composants explicités dans le fichier de configuration, les blocs intermédiaires, Blocs Système et les Bloc Globaux ainsi que tous les fils reliant ces instances.

Le fichier source de type HLL contient le programme correspondant à l'instanciation de tous les composants possédant une partie de leur modèle en langage de type HLL. Dans le cas de langages HLL orientés objet, le code contient les constructeurs d'objets de classes HLL avec les paramètres correspondants spécifiés dans la description de chaque composant. Un

tableau de formatage (TFMT) spécifique à chaque projet permet la génération du code de type HLL approprié.

Nous allons décrire le fichier de description d'architecture (FDARCH) pour une implémentation spécifique du système configurateur (PROGCONF) en langage PERL, choisi à cause de la manipulation directe des expressions régulières et des tables associatives à plusieurs niveaux.

L'architecture, représentée sur la figure 1, est constituée d'un processeur (CPU) communicant par une passerelle (BRIDGE) avec une mémoire système (MEMORY) et des entrées sorties (I/O).

Le fichier FDARCH représenté a été découpé pour faciliter son écriture et sa manipulation en plusieurs parties présentées dans les annexes A1 à A5. Ce fichier définit l'architecture du modèle global de simulation correspondant au schéma générique représenté sur la figure 1. Le langage de type HDL généré est VERILOG et le langage de haut niveau (HLL) généré est le C++.

Les propriétés des Composants (type, Interfaces de type HDL, ports, constructeurs d'objets de la classe C++, etc...) sont décrites dans plusieurs tables écrites en langage PERL. La correspondance entre ces tables et le diagramme illustrant le système Configurateur de la figure 2 est représentée sur la figure 6. Dans la suite de la description, toutes les notations commençant par "%..." correspondent à des tables de hachage.

Ces tables peuvent contenir la description, soit sous forme de suites de paramètres, soit sous forme de références à des procédures qui génèrent les valeurs requises.

Les procédures font partie de la description et sont changées pour chaque projet. L'exemple typique de leur utilisation est la génération d'interconnexion entre les composants. Souvent, le tableau correspondant à toutes les interconnexions possibles serait trop grand et difficile à générer et maintenir. En revanche une procédure peut être très compacte.

La description des règles d'interconnexion des composants (TCRC) contient des paramètres globaux communs à tous les types de composants.

Elle peut se présenter sous forme d'au moins une table. Dans le mode de réalisation de la figure 6, elle se présente sous forme de sept tables associatives (hachages), dont trois ont pour entrées (%InstNameModuleMap, %SysConnectMap, %SysSpecMap) les noms désignant l'ensemble des modèles possibles pour un même composant :

- La table "%InstNameModuleMap" (cf. 2 de A1), qui figure dans le fichier patent_config_defs.pm, pour décrire les Composants explicites. Cette table représente les règles de génération de connexion entre les signaux. Elle a pour entrées Connect et SelfConnect.
- La table "%SysConnectMap" qui figure dans le fichier patent_Sysconfig_defs.pm pour les Blocs Système.
- La table "%SysSpecMap" (cf. 4 de A2) qui figure dans le fichier patent_Sysconfig_defs.pm pour les Blocs Intermédiaires.

Les deux tables de hachage %SysWrapMap et %SysPinConst (cf. 6 de A2) contiennent les règles de connexion niveau système. La table de hachage %Activity_TypeMap associe le nom d'un composant avec son type. La table de hachage %PortProbeMap a pour entrées les noms des modules de type HDL utilisée dans la description des composants.

Le système Configurateur est piloté par au moins une table de règles de cohérence de connexions (TRCOH) qui représente les règles d'interconnexion entre les composants et d'insertion des composants intermédiaires. Dans le mode de réalisation de la figure 6, les règles de cohérence de connexions sont réparties sous forme des deux tables suivantes :

- %HwfConnectivityMap (cf. 2 de A2)
- %HwifAlternateMap

Le système Configurateur est piloté par au moins une table de formatage de fichiers source (TFMT) qui représente les règles de génération des instances des objets de type HLL. Dans le mode de réalisation de la figure 6, les règles de génération des instances des objets de type HLL sont réparties sous forme des deux tables suivantes :

- %moduleToCppClassMap (cf. 1 de A5)
- %classCppProtoMap (cf. 2 de A5)

Le rôle de chacune de ces tables de hachage est détaillé ci-après.

Parmi les entrées définies pour les tables %InstNameModuleMap, %SysConnectMap, %SysSpecMap, certaines sont obligatoires :

- L'entrée "MatchExpr", dont la valeur est l'expression régulière qui permet d'accepter un nom (label) du composant dans le fichier de configuration, est utilisée pour définir les variantes de positions autorisées pour le composant.
- L'entrée "ExtrExpr", dont la valeur est l'expression régulière qui assure l'extraction des paramètres numériques à partir du nom (label), est utilisée pour l'instanciation des Blocs Intermédiaires ou des Blocs Système ou pour générer le label du composant implicite.

D'autres paramètres sont facultatifs et peuvent être ajoutés à la description des tables mais ces paramètres ne peuvent être utilisés que par le code fourni avec la description – parmi eux :

- NumAdd est un paramètre auxiliaire servant de modificateur pour les paramètres extraits par ExtrExpr, exploité par la procédure GenDest (voir ci-après).
- BaseIdExpr est une expression régulière qui permet de retrouver le composant actif correspondant à un composant intermédiaire ou système en cas d'ambiguïté.

Les entrées suivantes du hachage définissent le Type du composant tel que, par exemple, DUT (modèle de type HDL), XACTOR (transactor), ou autres.

A chaque Type de Composant, correspond un hachage composé à son tour des entrées suivantes :

- L'entrée "ReqModule" - contient le nom du module de type HDL (VERILOG) du composant et le nom du fichier source correspondant.

- L'entrée "Connect" - est la définition de la méthode de sélection des signaux faisant partie d'un Port. Cette description est composée d'une suite d'entrés indexés par le nom du Port. À chaque nom de Port on associe un tableau d'expressions régulières et un pointeur sur une procédure de connexion des signaux qui gère l'application de ces expressions aux noms des signaux de l'interface du composant.

Le Configurateur dispose de plusieurs procédures préprogrammées mais d'autres peuvent être ajoutées et référencées. Les procédures préprogrammées acceptent des modificateurs :

- → E (exclusif – le signal ne sera utilise qu'une seule fois)
- → T(dérivation – le signal peut être utilisé par plusieurs destinations).
- → Une valeur spéciale 'filter_out_wire_name', attribué à un signal ou un groupe de signaux permet d'éliminer toute connexion sur les signaux correspondants.

- L'entrée "SelfConnect" – similaire à l'entrée "Connect" contient les définitions des connexions des signaux de Ports qui peuvent être connectés entre deux instances d'un même composant – cette définition facilite la connexion de signaux unidirectionnels sortie → entrée dans le cas particulier de connexion tête-bêche.

- L'entrée "Port" – fournit la définition de l'ensemble des ports. C'est un hachage qui associe le nom de chaque port à un tableau comportant les éléments suivants :
  - Le nom logique du composant de type HDL.
  - Le nom du module de type HDL ou un choix de modules.
  - Le numéro du port (utile pour la générations du code VERILOG et C++).

- L'entrée "GenDest" (cf. 1 de A3) – est un pointeur sur la procédure qui, pour un composant désigné par un label et type, génère, pour chaque port, le label du composant auquel il se connecte. La procédure référencée doit être impérativement spécifiée par

l'utilisateur – cette procédure est également utilisée pour contrôler la cohérence de la configuration demandée

- L'entrée "SysConn" – est un hachage qui associe à chaque module de type HDL contenu dans le modèle un pointeur sur la procédure de sélection et nommage des Blocs Système. La procédure référencée doit être impérativement spécifiée par l'utilisateur.
- L'entrée "GenHDLInstParam" - est un pointeur sur la procédure qui génère les paramètres supplémentaires requis par le simulateur de type HDL pour le code généré qui instancie la partie de type HDL du composant.
- L'entrée "CfgCpp" – définit les paramètres du constructeur de l'objet pour le code C++ d'identification de la Configuration, généré automatiquement, servant de filtrage pour la sélection des Composants par le système Configurateur :
  - Le premier paramètre est le nom de la classe de l'objet C++ – c'est un nom prédéfini associé à celui du modèle de type HDL – il est suivi par les tableaux correspondants à des groupes de paramètres associés typiquement à chaque port du composant. Le mot-clé ¨Own¨ indique un composant à modèle élémentaire.
  - Ensuite viennent le nom du composant de type HDL référencé dans la partie Port et l'identifiant de type du composant tels, par exemple, DUT, modèle du circuit en projet; XACTOR, transacteur.
  - Pour les blocs à modèle composé, la partie paramètres est plus riche :
    - → Elle contient le mot clé ¨Src¨ pour les composants actifs ou ¨Mon¨ pour les composants de Vérification et Monitoring
    - → Viennent ensuite les paramètres du composant adaptateur d'interface, comprenant :
      - le nom du composant de type HDL.
      - l'identifiant du type du composant et sa classe suivie du nom du port correspondant.

Ces derniers paramètres sont spécifiés pour chaque port du bloc composé.

Dans le code C++ généré pour un modèle composé, les paramètres transmis au constructeur d'une classe correspondent aux pointeurs sur les instances des adaptateurs d'interface.

Tous ces paramètres servent à guider le générateur du code C++ qui les combine avec les règles contenues dans le tableau de hachage "%classCppProtoMap"

- L'entrée "ProbeConnect" – associe à chaque port d'un composant Moniteur ou Vérifieur la classe C++ acceptable comme adaptateur d'interface. Ainsi le Configurateur est capable d'optimiser le code en utilisant le même adaptateur d'interface pour plusieurs composants (modèle composé).
- L'entrée "SysWrap" – est un hachage qui pour chaque partie de type HDL d'un composant définit l'expression régulière à utiliser pour mettre à un état connu les signaux restés non connectés après le câblage entre les composants.

L'algorithme de connexion de deux composants est le suivant :

- Le système Configurateur essaye d'abord la connexion directe.
- Si celle-ci s'avère impossible d'après la première table de hachage %HwfConnectivityMap, les modules spécifiés dans %HwifAlternateMap sont pris en compte pour être placés dans le composant. Les modules à choix sont indiqués par le caractère ¨>¨ au début de leur nom dans la spécification d'un composant.
- A la fin, si aucun module ne convient, le système Configurateur revient au hachage %HwfConnectivityMap pour choisir les blocs intermédiaires à placer entre les composants.
- Si aucun bloc adéquat n'est trouvé une erreur est signalée.

L'utilisateur peut influencer les choix du système Configurateur en spécifiant explicitement dans le fichier de configuration les blocs intermédiaires à connecter à un composant.

Le tableau "%HwfConnectivityMap" permet à la fois la Vérification de connectivité et la spécification de Blocs intermédiaires à insérer pour connecter deux Composants dont les ports ne sont pas directement connectables.

Pour que deux blocs puissent être connectés entre eux, il faut que %HwfConnectivityMap contienne des entrées qui décrivent les propriétés de cette connexion :

- A chaque composant, on associe le composant auquel il peut être connecté.
- Une connexion directe est indiquée par la valeur "Direct".
- Pour les connexions nécessitant un Bloc intermédiaire, le nom de ce bloc est indiqué.
- Pour certains blocs, la connectivité peut être spécifiée différemment pour chaque port du bloc. Cela sert à lever l'ambiguïté sur les connexions d'un bloc. Une utilisation typique concerne l'insertion de deux blocs intermédiaires connectés tête-bêche entre deux composants explicites.

On souligne que la connectivité est exprimée ici de manière globale, sans spécifier les noms de signaux à connecter.

Les Composants décrits en langage de type HLL peuvent posséder plusieurs implémentations possibles des modules de type HDL A_Port_i au niveau de chaque port i (voir Figure 3). La table de hachage "%HwifAlternateMap" spécifie le choix possible pour chaque module de type HDL d'un composant. Le système configurateur utilise cette information pour ne placer les blocs intermédiaires qu'en cas de stricte nécessité.

Le hachage %SysWrapMap définit, pour chaque implémentation d'un composant, son Bloc Système. Les entrées sont des noms de modules de type HDL associés avec le nom du bloc système et son implémentation de

type HDL. Ces informations sont utilisées par le système Configurateur pour instancier le bloc système adéquat à partir de spécifications contenues dans le hachage %SysConnectMap (cf. 3 de A2).

Le hachage %SysPinConst associe à chaque Bloc Système, les connexions de certains signaux de son interface aux noms des signaux (wires) correspondants au niveau système. Ces connexions sont indépendantes des configurations et ne sont pas régies par les expressions régulières applicables lors du câblage des autres signaux.

Typiquement, les signaux concernés sont des signaux d'horloge, de remise à zéro (reset), etc... Les noms référencés sont déclarés dans un code de type HDL, spécifique pour chaque modèle en projet (design), et fournis par l'utilisateur (sous forme d'une variable ou d'un fichier) avec les descriptions des composants et des règles de connexion spécifiques.

Ce code spécifique de type HDL est inclus systématiquement au début du code de type HDL généré par le système Configurateur.

La procédure "&gen_sys_VlogInstParameter" est une procédure qui génère les paramètres d'instantiation pour le générateur du code de type HDL à partir du label et du type de chaque Composant.

Le hachage %Activity_TypeMap (cf. 1 de A1) associe le nom d'un composant avec son type spécifié par un mot clé:

- 'actor' pour les composants actifs.
- 'spectator' pour les blocs de Monitoring.
- 'checker' pour les blocs de Vérification.
- 'helper' pour les blocs Système ou les blocs Intermédiaire en cas de connexion indirecte.

La table "%PortProbeMap" est un hachage dont les entrées sont les noms des modules de type HDL utilisés dans la description des composants. Les valeurs sont à leur tour des hachages qui associent, à chaque nom de port, un tableau contenant le nom de bloc pouvant servir de Sonde pour observer les signaux de ce port et le nom de la classe C++ correspondante. Ce hachage informe le système Configurateur sur le type de Composant capable d'observer les signaux d'un port donné et le type d'API C++ pouvant

être utilisée pour analyser le flux de contrôle et de données observé sur ce port.

La procédure &GenDest accepte en entrée le label du composant source, son type et son port. Le résultat retourné est un hachage qui associe le label du composant cible connecté avec son port.

Pour contrôler la cohérence de la configuration générée quel que soit l'ordre de prise en compte des Composants, le système Configurateur appelle la procédure &GenDest avec les paramètres obtenus pour le composant cible utilisé comme source et vérifie que le résultat correspond bien au composant et port source initial.

Le système Configurateur appelle la procédure &GenDest pour chaque composant contenu dans le fichier de configuration en parcourant tous ces ports.

Le hachage retourné par &GenDest peut contenir plusieurs composants avec leurs ports respectifs. C'est le cas notamment des connexions de type ¨bus¨ - &GenDest retourne tous les composants connectés par les bus sauf celui qui est donné en argument. Le système Configurateur insère automatiquement le composant « bus » nécessaire.

Le hachage %moduleToCppClassMap est un accélérateur de recherche permettant de retrouver la classe C++ à partir du nom de module de type HDL – il ne s'applique qu'aux Modèles élémentaires (voir 1.14).

Le hachage %classCppProtoMap décrit la génération des constructeurs d'objets de classes C++. Chaque entrée correspond au nom d'une classe C++. Chaque description est un hachage composé de quatre entrées :

- L'entrée ¨Prea¨ correspond à la génération de la première partie du constructeur (classe nom de l'instance, premiers paramètres).
- L'entrée ¨Iter¨ correspond à la partie itérative des paramètres du constructeur. Cette règle est appliquée pour chaque élément contenu dans l'entrée CfgCpp de la description du composant.

- L'entrée "Post" correspond à la partie finale du constructeur (derniers paramètres parenthèse fermante).
- L'entrée "Idle" est une règle de substitution pour la règle "Iter" pour les connexions non existantes (configurations partielles). Typiquement, un pointeur nul ou une chaîne nulle est généré(e).

Chaque entrée est une chaîne de caractères contenant des sélecteurs spéciaux prédéterminés, substitués par les valeurs calculées par le système Configurateur. Ces sélecteurs ont la forme générale #<Src|Dst|Act><Item># où 'Src', 'Dst' et 'Act' indiquent l'instance à prendre en compte , cette dernière étant transmise comme paramètre à un constructeur d'un objet HLL :

- 'Src' indique l'instance courante.
- 'Dst' l'instance connectée à l'autre extrémité du port.
- 'Act' indique l'instance composée correspondant au Composant Actif contenant le port d'observation.

<Item> est un mot-clé indiquant la sélection :

- 'Inst' pour le nom d'instance cpp.
- 'Iid' pour le label de composant correspondant à Inst.
- 'Ict' pour le type de composant (DUT, VERIFIER, XACTOR, Monitor...).
- 'Port' pour le nom du port.

Les deux variables "$cpp_preamble" (cf. 3 de A5), "$cpp_postambl" (cf. 4 de A5) contiennent respectivement le début et la fin du texte de programme C++ généré par le système Configurateur. Le texte généré à partir du tableau %classCppProtoMap est inséré au milieu.

Les deux variables "$top_verilog_preamble" et "$top_verilog_postamble" contiennent respectivement le début et la fin du texte de l'instance 'top' de la description de type HDL (VERILOG) de la

configuration. Typiquement, ce sont les instanciations des blocs d'horloge et autres blocs niveau système.

La description qui suit est un exemple simple de génération de Configuration par la mise en œuvre du système Configurateur qui est illustrée dans le cas simplifié ci-après de l'architecture d'un Système constitué d'un pont (BRIDGE) reliant une unité centrale (CPU), une Mémoire et une carte entrée/sortie (I/O). L'Architecture du Système est représentée sur la figure 1.

Dans cet exemple, le CPU et le BRIDGE peuvent être modélisés, soit par un modèle VERILOG du projet "Design" (DUT, DUT_Core), soit par un modèle composé C++ (XACTOR).

L'interface entre le CPU et le BRIDGE est nommée FBUS. Plusieurs variantes de cette interface sont considérées correspondant à des étapes successives de développement.

Le bloc global Clock assure la distribution des signaux d'horloge et signaux système globalement pour l'ensemble des Composants

Le système Configurateur s'appuie sur la description des Interfaces de type HDL des Composants. Des interfaces simples, écrites en langage VERILOG, sont proposées ci-dessous à titre d'illustration pour chacun des Composants. Le système Configurateur n'analyse que cette partie des fichiers source contenus dans la bibliothèque des fichiers sources de type HDL (BFSHDL). Ils s'inscrivent dans une méthodologie de prise en compte progressive de variantes de modèles au fur et à mesure de leur disponibilité.

La description de type HDL des Interfaces qui suit est réduite à ce qui est nécessaire à l'illustration des possibilités et du fonctionnement du configurateur. Dans une situation réelle sont inclues les descriptions des comportements comme dans le cas du modèle de l'horloge (module CLOCK).

La description est la suivante :

- CPU :

a) Le modèle DUT de composant CPU (figure 7a) présente un port FBUS de connexion avec le BRIDGE ainsi qu'un ensemble de signaux Système définis ci-après.

```
module cpu (
        XXadr_dat,
        XXreq,
        XXack,
        //
        clk,
        reset,
        powergood
        );
inout  [63:0] XXadr_dat;
inout  [3:0] XXreq;
inout  [3:0] XXack;

input  clk;
input  reset;
input  powergood;

endmodule
```

b) Le modèle XACTOR (figure 7b) consiste en un modèle abstrait C++ dont l'instance de type HDL est constituée de celle du Port nommé fbus_p. Dans cet exemple, il est supposé que le Port FBUS présente une variante nommée FBUSA dans laquelle les signaux bidirectionnels adr_dat ont été éclatés en paires de signaux unidirectionnels in (inXXadr_dat) et out (outXXadr_dat).

```
module fbus_p (
        inXXadr_dat,
        outXXadr_dat,
        inXXreq,
        outXXreq,
        inXXack,
        outXXack,
        //
        clk,
        reset,
        powergood
```

```
);

input  [63:0] inXXadr_dat;
output [63:0] outXXadr_dat;
input  [3:0] inXXreq;
output [3:0] outXXreq;
input  [3:0] inXXack;
output [3:0] outXXack;

input  clk;
input  reset;
input  powergood;

endmodule
```

c) Le modèle MONITOR du composant CPU est un modèle composé permettant le Monitoring de l'interface FBUS. Dans le cas où il n'existe pas d'adaptateur de port fbus_p dans la configuration, une sonde contenant l'instance fbus_probe sera générée (figure 7c).

```
module fbus_probe (
        XXadr_dat,
        XXreq,
        XXack,
        //
        clk,
        reset,
        powergood
        );
input  [63:0] XXadr_dat;
input  [3:0] XXreq;
input  [3:0] XXack;

input  clk;
input  reset;
input  powergood;

endmodule
```

- Bloc intermédiaires d'adaptation FBUS :

Ces blocs réalisent l'adaptation d'interface au niveau FBUS dans le cas où la correspondance point à point des signaux est impossible. Le port FBUSA est la variante de FBUS dans laquelle les signaux bidirectionnels ont été éclatés en paires de signaux unidirectionnels d'entrée (in) et de sortie (out).

Le modèle fbus_xchg (figure 7d) réalise l'adaptation entre l'interface FBUS du port fbus_p et celui d'un port FBUSA – la largeur de l'interface peut être paramétrée de façon à rendre compte d'évolutions technologiques possibles.

```
module fbus_xchg (
        XXadr_dat,
        XXreq,
        XXack,
        //
        inXXadr_dat,
        outXXadr_dat,
        inXXreq,
        outXXreq,
        inXXack,
        outXXack
);
inout  [63:0] XXadr_dat;
inout  [3:0] XXreq;
inout  [3:0] XXack;
        //
input  [63:0] inXXadr_dat;
output [63:0] outXXadr_dat;
input  [3:0] inXXreq;
output [3:0] outXXreq;
input  [3:0] inXXack;
output [3:0] outXXack;

// model body

endmodule
```

- BRIDGE :
  a) Le modèle DUT du composant BRIDGE (figure 7e) présente les trois ports respectifs vers les modèles CPU, MEM et IO ainsi que les

signaux Système délivrés par le bloc système dédié SYS_BRIDGE et par le bloc global CLOCK.

```
module bridge (
        XXadr_dat,
        XXreq,
        XXack,
        YYadr,
        YYdata,
        ZZdata,
        ZZreq,
        ZZack,
        YYctrl,
        clk_2xp,
        clk_2xn,
        clkp,
        clkn,
        reset,
        powergood
        );
inout   [63:0] XXadr_dat;
inout   [3:0] XXack;
inout   [3:0] XXreq;

output [31:0] YYadr;
inout   [63:0] YYdata;
output [2:0] YYctrl;

inout   [15:0] ZZdata;
input   [1:0] ZZack;
output [1:0] ZZreq;

input   clk_2xp;
input   clk_2xn;
input   clkp;
input   clkn;
input   reset;
input   powergood;

endmodule
```

b) Le modèle BRIDGE_CORE (figure 7f) est la variante DUT_CORE du composant BRIDGE présentant une interface FBUSA. Ce modèle traduit la situation où le modèle du contrôleur d'interface FBUS n'est pas disponible.

```
module bridge_core (
        inXXadr_dat,
        outXXadr_dat,
        inXXreq,
        outXXreq,
        inXXack,
        outXXack,
        //
        YYadr,
        YYdata,
        YYctrl,
        //
        ZZdata,
        ZZreq,
        ZZack,
        //
        clk_2xp,
        clk_2xn,
        clkp,
        clkn,
        reset,
        powergood
        );
input  [63:0] inXXadr_dat;
output [63:0] outXXadr_dat;
input  [3:0] inXXreq;
output [3:0] outXXreq;
input  [3:0] inXXack;
output [3:0] outXXack;

output [31:0] YYadr;
inout  [63:0] YYdata;
output [2:0] YYctrl;

inout  [15:0] ZZdata;
input  [1:0] ZZack;
output [1:0] ZZreq;

input  clk_2xp;
input  clk_2xn;
input  clkp;
input  clkn;
input  reset;
input  powergood;
```

```
endmodule
```

c) Le modèle sys_bridge (figure 7g) est le block système dédié aux signaux Système du modèle Bridge – il est directement alimenté par le bloc CLOCK.

```
module sys_bridge(
                clk_2xp,
                clk_2xn,
                clkp,
                clkn,
                reset,
                powergood,
                //
                sys_CLK_2X,
                sys_CLK,
                sys_RESET,
                sys_POWER_GOOD
                );

output          clk_2xp;
output          clk_2xn;
output          clkp;
output          clkn;
input           sys_CLK_2X;
input           sys_CLK;
input           sys_RESET;
input           sys_POWER_GOOD;
output          reset;
output          powergood;

clk_2xp = sys_CLK_2X;
clk_2xn = ~sys_CLK_2X;
clkp    = sys_CLK;
clkn    = ~sys_CLK;
reset   = sys_RESET;
powergood = sys_POWER_GOOD;

endmodule
```

d) Le modèle XACTOR (figure 7h) est un modèle abstrait C++ dont l'instance de type HDL est constituée de celle des différents ports respectivement fbus_p, cmem_p et cio_p vers CPU, MEM et IO.

```
module cmem_p (
        YYadr,
        YYdata,
        YYctrl,
        clk,
        reset,
        powergood
        );
output [31:0] YYadr;
inout  [63:0] YYdata;
output [2:0] YYctrl;

input  clk;
input  reset;
input  powergood;

endmodule

module cio_p (
        ZZdata,
        ZZreq,
        ZZack,
        //
        clk,
        reset,
        powergood
        );
inout   [15:0] ZZdata;
input  [1:0] ZZack;
output [1:0] ZZreq;

input  clk;
input  reset;
input  powergood;

endmodule
```

– MEMORY :

Le modèle DUT du composant MEMORY (figure 7i) présente les ports respectifs vers les modèles BRIDGE et CLOCK.

modèle DUT :

```
module cmem (
        YYadr,
        YYdata,
        YYctrl,
```

```
                clk,
                reset,
                powergood
                );
        input   [31:0] YYadr;
        inout   [63:0] YYdata;
        input   [2:0] YYctrl;

        input   clk;
        input   reset;
        input   powergood;

        endmodule
```

- I/O :

Le modèle DUT du composant I/O (figure 7j) présente les ports respectifs vers les modèles BRIDGE et CLOCK.

modèle DUT :

```
        module cio (
                ZZdata,
                ZZreq,
                ZZack,

                clk,
                reset,
                powergood
                );
        inout   [15:0] ZZdata;
        output  [1:0] ZZack;
        input   [1:0] ZZreq;

        input   clk;
        input   reset;
        input   powergood;

        endmodule
```

- Block Système global :

Ce modèle (figure 7k) assure la fourniture des signaux Système alimentant les blocs système dédiés à chaque composant.

```
        module Clock (
                                sys_POWER_GOOD,
```

```
                                    sys_RESET,
                                    sys_CLK,
                                    sys_CLK_2X
                                    );
    output sys_POWER_GOOD;
    output sys_RESET;
    output sys_CLK;
    output sys_CLK_2X;

    parameter HALF_PERIOD = 75; // 7.5ns, 66MHz
    parameter RESET_DELAY = 10000000000 ; // 1.0ms
    parameter POWER_GOOD_DELAY = 5000000000 ; // 0.5ms
    initial begin;
     sys_POWER_GOOD = 0;
    sys_ RESET = 0;
     sys_CLK = 0;
     sys_CLK_2X = 0;
     #POWER_GOOD_DELAY sys_POWER_GOOD = 1;
     #RESET_DELAY sys_RESET = 1;
    end

    always begin
     #HALF_PERIOD sys_CLK = ~sys_CLK;
    end
    always @(posedge sys_CLK_2X) begin
      sys_CLK = ~sys_CLK;
    end

    endmodule
```

Définition des configurations :

Les variantes de configuration marquent les différentes étapes de vérification suivant la disponibilité des modèles – le système Configurateur élabore les configurations de façon non ambiguë à partir des informations topologiques contenues dans le fichier de configuration.

La première configuration est définie par le fichier de configuration 1 ci-après et représentée figure 8a : les modèles DUT du CPU et du BRIDGE ne sont pas disponibles.

Fichier de configuration 1 :

```
CPU_0      XACTOR
CPU_0      MONITOR
BRIDGE_0   XACTOR
CMEM_0     DUT
CIO_0      DUT
```

Le XACTOR CPU partage le port fbus_p avec le MONITOR.

La deuxième configuration est définie par le fichier de configuration 2 ci-après et représentée figure 8b.

Le BRIDGE XACTOR est connecté au CPU XACTOR par une interface de type FBUSA spécifiée dans le fichier de configuration par l'attribut FBUSA=FBUS_xchg – le système Configurateur insère automatiquement le deuxième adaptateur fbus_xchg 102.

Fichier de configuration 2 :

```
CPU_0      XACTOR    FBUSA=FBUS_xchg
CPU_0      MONITOR
BRIDGE_0   XACTOR
CMEM_0     DUT
CIO_0      DUT
```

Le XACTOR CPU partage le port fbus_p avec le Monitor.

La troisième configuration est définie par le fichier de configuration 3 ci-après et représentée figure 8c : le BRIDGE DUT_CORE est connecté au CPU XACTOR par une interface de type FBUSA spécifiée dans le fichier de configuration par l'attribut FBUSA=FBUS_xchg – le système Configurateur insère automatiquement le deuxième adaptateur fbus_xchg 102.

Fichier de configuration 3 :

```
CPU_0      XACTOR    FBUSA=FBUS_xchg
CPU_0      MONITOR
BRIDGE_0   DUT_CORE
```

```
CMEM_0    DUT
CIO_0     DUT
```

Le XACTOR CPU partage le port fbus_p avec le MONITOR.

La quatrième configuration est définie par le fichier de configuration 4 ci-après et représentée figure 8d : le modèle CPU DUT est connecté avec le modèle BRIDGE XACTOR – le système Configurateur insère automatiquement un adaptateur d'interface fbus pour réaliser la connexion entre le CPU DUT et le BRIDGE XACTOR sans mention explicite de ce bloc dans le fichier de configuration.

Fichier de configuration 4 :

```
CPU_0     DUT
CPU_0     MONITOR
BRIDGE_0  XACTOR
CMEM_0    DUT
CIO_0     DUT
```

Le Configurateur instancie automatiquement la Sonde fbus_probe pour permettre l'observation de l'interface FBUS par le Monitor.

La cinquième configuration est définie par le fichier de configuration 5 ci-après et représentée figure 8e : tous les modèles DUT sont disponibles.

Fichier de configuration 5 :

```
CPU_0     DUT
CPU_0     MONITOR
BRIDGE_0  DUT
CMEM_0    DUT
CIO_0     DUT
```

Le système Configurateur instancie automatiquement la Sonde fbus_probe pour permettre l'observation de l'interface FBUS par le Monitor.

La sixième configuration est définie par le fichier de configuration 6 ci-après et représentée par la figure 8f : c'est une Configuration Multi-serveur - la Config1 est répartie entre les 2 serveurs SERVER1 et SERVER2, la coupure se faisant au niveau de l'interface FBUS.

Fichier de configuration 6 :

```
CPU_0      XACTOR    SERVER1
CPU_0      MONITOR   SERVER1
BRIDGE_0   XACTOR    SERVER2
CMEM_0     DUT       SERVER2
CIO_0      DUT       SERVER2
```

Les règles de connectivité permettant la génération des tables de connectivités et de toute configuration pour cet exemple sont décrites en langage PERL dans les annexes A1 à A5.

L'annexe A1 décrit, dans le cas de l'exemple choisi, les règles topologiques liées aux composants actifs à utiliser par le système Configurateur.

L'annexe A2 décrit les règles topologiques liées aux Blocs systèmes et aux Blocs intermédiaires à utiliser par le système Configurateur.

L'annexe A3 décrit les procédures de connexion des Composants à utiliser par le système Configurateur.

L'annexe A4 décrit le code de génération du fichier en langage de type HDL (Verilog_top.v) de connexion des composants à utiliser par le système Configurateur.

L'annexe A5 décrit le code de génération des Classes C++.

Les annexes A6 et A7 décrivent les fichiers résultats générés par le système Configurateur respectivement en langage bas niveau de type HDL (VERILOG) et en langage haut niveau (C++) pour constituer le modèle de la configuration définie par la troisième configuration correspondant à la figure 8c. Le configurateur peut ainsi, avec l'utilisateur-développeur, réaliser son modèle selon la variante de configuration choisie.

On aurait pu, de la même façon, illustrer les autres variantes de réalisation du modèle (figures 8a, 8b et 8d à 8f) qui pourraient être traitées par le système Configurateur pour produire les fichiers résultats correspondants, constituant les modèles de simulation de l'architecture système envisagée pour le circuit intégré en développement.

La figure 2 représente les divers moyens mis en œuvre par un système informatique 40 mettant en œuvre le procédé de l'invention.

Le calculateur 40 comporte une unité centrale 41, comprenant des circuits de traitement de données 42 (unité de traitement arithmétique ALU et mémoires de programmes associés, globalement appelés, par la suite, ALU, par commodité), et comporte diverses mémoires, de programmes ou fichiers de données d'exploitation, associées à l'unité centrale 41.

L'unité centrale 41 comporte une mémoire 51 contenant la table de composants et de règles de connexions (TCRC), la table de règles de cohérence de connexions (TRCOH) et la table de formatage de fichiers source (TFMT). Ces tables sont créées à partir du fichier de description d'architecture (FDARCH).

La mémoire 51 contient aussi des programmes ou moteurs intelligents PROGCONF nécessaires pour constituer le système Configurateur exploitable par l'ALU 42.

Une mémoire 53 contient la table de connexion des instances (TCINST) représentant les instances des composants et leurs interconnexions mutuelles requises par la Configuration définie dans le fichier de configuration FCONF et conformes au règles contenues dans les tables TCRC et TRCOH.

La mémoire 53 contient aussi la table de câblage (TCAB) représentant les connexions physiques (fils) entre les parties de type HDL des composants instanciés à partir de la bibliothèque de fichiers sources de type HDL (BFSHDL).

Il en résulte les fichiers MGHDL et MGHLL représentant respectivement les fichiers sources du modèle global de simulation de type HDL (VERILOG ou VHDL) et de type HLL (C++) générés par le système

Configurateur, modèle permettant la co-simulation d'un circuit ASIC dans son environnement de vérification.

La mise en œuvre du procédé de l'invention va maintenant être exposée.

Le procédé d'établissement automatique, au moyen du calculateur ou système de traitement de données 40, de modèle de simulation d'une configuration de modèles de composants déterminés, mutuellement reliés par des connexions d'inter-fonctionnement, pour constituer un modèle global de simulation MGHDL / MGHLL d'un circuit ASIC dans son environnement de vérification, comporte les étapes suivantes (cf. figure 2a) :

- un opérateur, éventuellement aidé ou même remplacé par un calculateur, établit le fichier de description d'architecture (FDARCH) décrivant les règles d'interconnexion entre les différents composants, les modèles pouvant être utilisés pour modéliser chacun de ces composants et les règles de formatage pour générer les fichiers source du modèle résultant.
- un opérateur, éventuellement aidé ou même remplacé par un calculateur, établit la bibliothèque BFSHDL de fichiers source, de parties de type HDL de modèles de composants respectifs susceptibles d'être utilisés dans une configuration conformément au contenu du fichier FDARCH.
- l'opérateur établit le fichier FCONF de configuration visée, identifiant les composants et les modèles voulus (cf. figure 2b).
- Le système de traitement lit le fichier de description d'architecture (FDARCH) généré et génère et place en mémoire les tables TCRC, TRCOH et TFMT à partir de FDARCH (cf. figure 2b).
- Les diverses tables TCRC, TRCOH et TFMT étant rendues accessibles au système de traitement de données 40 (cf. figure 2c),

- le système de traitement de données 40 lit le fichier de la configuration visée FCONF pour identifier les composants et leurs modèles requis (cf. figure 2c),
- il lit la table de composants et de règles de connexion pour instancier et placer dans la table de connexion des instances TCINST les composants requis (cf. figure 2c).
- Il lit la table de règles de cohérence de connexions (TRCOH) pour vérifier la connectivité physique entre modèles et insérer, le cas échéant, des composants intermédiaires.
- Il lit dans la bibliothèque BFSHDL les fichiers source correspondants aux parties de type HDL des modèles de composants instanciés dans la table TCINST pour engendrer la table TCAB de connexions physiques (fils) entre les modèles de composants (cf. figure 2c).
- Il applique les règles de formatage de la table TFMT au contenu des tables TCINST et TCAB et il génère les fichiers source finaux MGHDL/MGHLL de modèle global de simulation correspondant à la configuration spécifiée par le fichier de configuration (FCONF) (cf. figure 2d).

Le fichier d'entrée FCONF correspond donc à une description du schéma théorique du type de la figure 1, sans toutefois les modèles, qui sont ajoutés lors de l'élaboration des tables intermédiaires TCINST, TCAB descripteurs de configuration et de connexions, afin de fournir un modèle global de simulation qui soit effectivement opérationnel et qui, en outre, ici, permette une observation de signaux particuliers.

Les diverses tables ci-dessus peuvent être préalablement stockées dans le calculateur 40 ou bien avoir été stockées dans une mémoire externe que l'on relie au calculateur 40, éventuellement par un réseau de transmission de données pour qu'elles lui soient accessibles.

L'ALU 42 dispose alors de toutes les données de base nécessaires aux tables outils TCRC, TRCOH, TFMT pour que le programme PROGCONF associé au système Configurateur, commande, à l'ALU 42, l'élaboration des fichiers finaux MGHDL/MGHLL (cf. figure 2d).

La table intermédiaire TCAB, des descripteurs de connexions physiques, générée ici à partir d'une description de type HDL, est, dans cet exemple, soumise à une vérification, afin de détecter des erreurs manifestes, telles que l'existence d'entrées ou sorties non raccordées ou encore de sorties reliées en parallèle, lorsqu'il s'agit de sorties classiques de type dit 'totem', donc autres que celles à collecteur ouvert ou à inhibition commandée (tri-state).

En cas de modification du fichier de départ FCONF, les tables intermédiaires TCINST, TCAB et les fichiers finaux MGHDL/MGHLL peuvent ainsi être redéfinis à partir des fichiers sources de modèles et de fichier FDARCH. On évite ainsi des risques d'erreur.

Dans cet exemple, l'ALU 42 génère un modèle global de co-simulation en associant, à au moins un bloc fonctionnel, plusieurs modèles de spécification fonctionnelle écrits en langages de programmation de niveaux différents, ici de type HDL et C++.

On conçoit que la présente invention peut être mise en œuvre selon d'autres formes spécifiques, sans sortir de son domaine d'application tel que revendiqué. Par conséquent, la présente description détaillée doit être considérée comme étant une simple illustration d'un cas particulier dans le cadre de l'invention et peut donc être modifiée sans sortir du domaine défini par les revendications jointes.

ANNEXE : Description du code Configurateur

A1 - Règles Topologiques liées aux Composants actifs

```
#! /usr/triton/bin/perl
#####################################################################
##################
#
#       Copyright (c) 2000 BULL - Worldwide Information Systems
#                          All rights reserved
#
#  Module name   :  patent_config_defs.pm
#  Author        :  Andrzej Wozniak
#
#####################################################################
##################

###
%Activity_TypeMap =                                    1 de A1
  (
   "XACTOR"              => "actor",
   "DUT"                 => "actor",
   "DUT_CORE"            => "actor",
   "MONITOR"             => "spectator",
   "IND_CON"             => "helper",
   "SYS_CON"             => "helper",
   );


###################
%InstNameModuleMap =                                   2 de A1
  ###################
  (
   #######
   "CPU" =>                  liste des labels CPU acceptés
   #######
   { "MatchExpr" => "^(CPU)(?:_[0-3])\$" ,
     "ExtrExpr"  => "^CPU_([0-3])\$" ,
     ####
     "DUT"  =>
     { "ReqModule" => { "cpu" => "cpu_model.v",
             },
                      expressions régulières de connexion
       "Connect" => { FBUS => [\&subst_infix,  ["(XX.*)",
""],],
             },
       "Port"    => { FBUS => [CPU_a, cpu, 0], },
```

```
        "genDest" => \&genDest,
        "SysConn" => { CPU_a => \@std_glob_con,
                },
        "CfgCpp"  => [CPU_Dut, [ ['Own', [CPU_a , DUT, ],
['None'], ], ], ],

    },
```

CPU_a = nom logique
cpu = nom de l'instance Verilog générée
0 = n° de port
\&genDest = définit la procédure générant le genDest

```
    "XACTOR"  =>
    { "ReqModule" => { "fbus_p" => "fbus_port.v",
                },
```

même nom de connexion de part et d'autre

```
      "Connect"     => { "FBUSA"  => [\&subst_infix,
["in(.*)", "aa_con"], ["out(.*)", "bb_con"],],
                },
```

Connexion tête-bêche

```
      "SelfConnect" => { "FBUSA"  => [\&subst_infix,
["out(.*)", "aa_con"], ["in(.*)", "bb_con"],],
                },
      "Port"    => { FBUSA => [FBUS_p, fbus_p, 0], },
      "genDest" => \&genDest,
      "SysConn" => { FBUS_p => \@std_glob_con,
                },
      "CfgCpp"  => [CPU_Xactor, [ ['Src', [FBUS_p ,
FBUS_type, Fbus_hwif], [FBUS] ] ] ],

    },
    "MONITOR"  =>
    { "ReqModule" => { "fbus_p" => "fbus_port.v",
                },

      "Connect"     => { "FBUS"  => [\&subst_infix,
["in(.*)", "aa_con"], ["out(.*)", "bb_con"],],
                },
      "SelfConnect" => { "FBUS"  => [\&subst_infix,
["out(.*)", "aa_con"], ["in(.*)", "bb_con"],],
                },
      "Port"    => { FBUS => [FBUS_p, fbus_p, 0], },
```

```
          "genDest" => \&genDest,
          "SysConn" => { FBUS_p => \@std_glob_con,
                 },
          "ProbeConnect" => { FBUS => Fbus_hwif,
                    },
          "CfgCpp"  => [CPU_Monitor, [ ['Mon', [FBUS_p ,
    FBUS_type, Fbus_hwif], [FBUS],],],],

        },
         ####
       },### CPU
       ######
       "BRIDGE" =>
       ######
       {
         "MatchExpr" => "^(BRIDGE)(?:_[0-1])\$" ,
         "ExtrExpr"  => "^BRIDGE_([0-1])\$" ,
         ####
         "DUT"  =>
         { "ReqModule" => {"bridge" => "bridge_x.v",
                 },
           "Connect" => { FBUS => [\&subst_infix,  ["(XX.*)",
    ""],],
                    CMEM => [\&subst_infix,  ["(YY.*)",
    ""],],
                    CIO  => [\&subst_infix,  ["(ZZ.*)",
    ""],],
                },

          "Port"    => {  FBUS => [BRD, bridge, 0],
                          CMEM => [BRD, bridge, 1],
                          CIO  => [BRD, bridge, 2],
                 },

          "genDest" =>  \&genDest,
          "SysConn" => { BRD => [\&getSpecSysInst, "E",
    [\&subst_infix,  ["(.*)", ""],],],
                },
          "CfgCpp"  => [Bridge_Dut, [ ['Own', [BRD, DUT, ],
    ['None'], ],
                              ],
                    ],
         },
          ####
         "DUT_CORE"  =>
         { "ReqModule" => {"bridge_core" => "bridge_core.v",
                 },
```

```
        "Connect" => { FBUSA => [\&subst_infix,
    ["(XX.*)", ""],["in(.*)", "aa_con"], ["out(.*)",
    "bb_con"],],
                    CMEM  => [\&subst_infix,  ["(YY.*)",
    ""],],
                    CIO   => [\&subst_infix,  ["(ZZ.*)",
    ""],],
                },

        "SelfConnect" => { "FBUSA"  => [\&subst_infix,
    ["out(.*)", "aa_con"], ["in(.*)", "bb_con"],],
                    },

         "Port"    => {  FBUSA => [BRD, bridge_core, 0],
                    CMEM  => [BRD, bridge_core, 1],
                    CIO   => [BRD, bridge_core, 2],
                },

         "genDest" =>  \&genDest,
         "SysConn" => { BRD => [\&getSpecSysInst, "E",
    [\&subst_infix,  ["(.*)", ""],],],
                },
         "CfgCpp"  => [Bridge_Dut, [ ['Own', [BRD,
    DUT_CORE, ], ['None'], ],
                            ],
                ],
      },
       ####
      "XACTOR"  =>
      { "ReqModule" => { "fbus_p" => "fbus_port.v",
                    "cmem_p" => "cmem_port.v",
                    "cio_p"  => "cio_port.v",
                },
        "Connect" => { FBUSA => [\&subst_infix,  ["in(.*)",
    "aa_con"], ["out(.*)", "bb_con"],],
                  CMEM  => [\&subst_infix,  ["(YY.*)",
    ""],],
                  CIO   => [\&subst_infix,  ["(ZZ.*)",
    ""],],
                },

        "SelfConnect" => { "FBUSA"  => [\&subst_infix,
    ["out(.*)", "aa_con"], ["in(.*)", "bb_con"],],
                    },

        "Port"    => {  FBUSA => [FBUS_p, fbus_p, 0],
                  CMEM  => [CMEM_p, cmem_p, 1],
                  CIO   => [CIO_p,   cio_p, 2],
```

```
                },

        "genDest" =>  \&genDest,
        "SysConn" => { FBUS_p => \@std_glob_con,
                CMEM_p => \@std_glob_con,
                CIO_p  => \@std_glob_con,
               },
        "CfgCpp"  => [Bridge_Xactor, [ ['Src', [FBUS_p,
FBUS_type, Fbus_hwif], [FBUS] ],
                           ['Src', [CMEM_p, CMEM_type,
Cmem_hwif], [CMEM] ],
                           ['Src', [CIO_p,  CIO_type,
Cio_hwif], [CIO] ],
                           ],
                ],
      },
      },
     ####
     ######
    "CIO" =>
     ######
     {
       "MatchExpr" => "^(CIO)(?:_[0-1])\$" ,
       "ExtrExpr"  => "^CIO_([0-1])\$" ,
       ####
       "DUT"  =>
       { "ReqModule" => { "cio" => "cio_model.v",
                 },
         "Connect" => { CIO  => [\&subst_infix,  ["(ZZ.*)",
""],],
                },

         "Port"    => {  CIO  => [CIOD,   cio, 0],
                 },

         "genDest" =>  \&genDest,
         "SysConn" => { CIOD => \@std_glob_con,
                },
         "CfgCpp"  => [Cio_Dut, [ ['Own', ['Src', [CIOD,
DUT,], [None] ],],
                     ],
                 ],
      },
        ####
     "XACTOR" =>
     { "ReqModule" => { "cio_p" => "cio_model.v",
                },
```

```
        "Connect" => { CIO  => [\&subst_infix,  ["(ZZ.*)",
    ""],],
                },

        "Port"    => {  CIO  => [CIOD,   cio, 0],
                  },

        "genDest" =>  \&genDest,
        "SysConn" => { CIOD => \@std_glob_con,
                  },

        "CfgCpp"  => [Cio_Dut, [ ['Own', ['Src', [CIOD,
    DUT,], [None] ],],
                       ],
                  ],
         },
         },
       ######
      "CMEM" =>
       ######
       {
         "MatchExpr" => "^(CMEM)(?:_[0-1])\$" ,
         "ExtrExpr"  => "^CMEM_([0-1])\$" ,
         ####
         "DUT"  =>
         { "ReqModule" => { "cmem" => "cmem_model.v",
                  },
           "Connect" => { CMEM  => [\&subst_infix,
    ["(ZZ.*)", ""],],
                },

         "Port"    => {  CMEM  => [CMEMD,   cmem, 0],
                  },

         "genDest" =>  \&genDest,
         "SysConn" => { CMEMD => \@std_glob_con,
                  },

         "CfgCpp"  => [Cmem_Dut, [ ['Own', ['Src', [CMEMD,
    DUT,], [None] ],],
                       ],
                  ],
       },
         ####
      "XACTOR" =>
       { "ReqModule" => { "cmem_p" => "cmem_model.v",
                  },
```

```
        "Connect" => { CMEM  => [\&subst_infix,  ["(ZZ.*)",
    ""],],
                },

        "Port"    => {  CMEM  => [CMEMD,   cmem, 0],
                  },

        "genDest" =>  \&genDest,
        "SysConn" => { CMEMD => \@std_glob_con,
                },
        "CfgCpp"  => [Cmem_dut, [ ['Own', ['Src', [CMEMD,
    DUT,], [None] ],],
                        ],
                  ],
         },
         }
    );

    %PortProbeMap =                                    3 de A1
    (
     'cpu'=> {
            "FBUS"  => ["FBUS_PROBE" , Fbus_hwif ],
           },
    );
    ######
    1; ### makes perl happy
```

ANNEXE

A2 - Règles Topologiques liées aux Blocs système et blocs intermédiaires

```
#! /usr/triton/bin/perl
#############################################################
######################
#
#      Copyright (c) 2000 BULL - Worldwide Information
Systems
#                          All rights reserved
#
#  Module name   :  patent_sys_config_defs.pm
#  Author        :  Andrzej Wozniak
#
#  Description   : tables for driving configuration
#           generation aux part
#
#  Rev Date : $Date: 2001-03-06 19:11:20+01 $
#
#############################################################
######################
#########
$GlobSysInst          = 'SysClock';
$GlobSysInstModule    = 'Clock';
$GlobSysInstModuleSrc = 'patent_sys_glob.v';

# Factorisation de l'expression commune à l'ensemble des
connexions aux Blocs Globaux
@std_glob_con = (\&getGlobSysInst, "E",
            [\&subst_infix,
             ["^(?:clk)\$",  "sys_CLK_2X"],
             ["^(?:reset)\$",  "sys_RESET"],
             ["^(?:powergood)\$",  "sys_POWER_GOOD"],
             ],);

#######
%HwfConnectivityMap =
  (
   "bridge" => {
     "cpu"     => "Direct",
     "cmem"    => "Direct",
     "cio"     => "Direct",
     "fbus_p"  => "fbus_xchg",
     "fbus_xchg" => { FBUS => 'Direct',
              },
   },
```

1 de A2

2 de A2

```
      "bridge_core" => {
        "cpu"      => "fbus_xchg",
        "cmem"     => "Direct",
        "cio"      => "Direct",
        "fbus_p"   => "Direct",
       "fbus_xchg" => { FBUSA => 'Direct', ,
              FBUS  => 'fbus_xchg',
                 },
      },

      "fbus_p"   => {
        "fbus_xchg" => { FBUSA => 'Direct',
                   FBUS => 'fbus_xchg',
                 },
   "cpu"       => "fbus_xchg",
        "fbus_p"     => 'Direct',
      },

      "cio"     => { "cio_p"   => 'Direct' },
      "cmem"    => { "cmem_p" => 'Direct' },
      "fbus_xchg" => { "fbus_xchg" => { FBUS => 'Direct' },
     "cpu"        => { "fbus_xchg" => { FBUS => 'Direct' },

               },
    );
#######
%HwifAlternateMap =
(
 );
###
%SysWrapMap =
(
 bridge         => [ BRIDGE_sys,        sys_bridge  ],
 bridge_core    => [ BRIDGE_sys,        sys_bridge  ],
);

#######
%SysConnectMap =
  (
    ######
    "$GlobSysInst" =>
    ######
    { "MatchExpr" => "^($GlobSysInst)\$",
      "ExtrExpr"  => "^($GlobSysInst)\$",
      ####
      "SYS_CON"   =>
      {"ReqModule" => {"$GlobSysInstModule" =>
"$GlobSysInstModuleSrc",
               },
```

3 de A2

```
        "Connect"    => { "Global" => [\&subst_infix,
["(.*)", ""], ],
                    },
      },
    }, #### $GlobSysInst
    ######
    "BRIDGE_sys" =>
    ######
    { "MatchExpr" => "^(?:BRIDGE_[0-3]_)(sys)\$",
      "ExtrExpr"  => "^BRIDGE_([0-3])_sys\$",
      ####
      "SYS_CON"   =>
      { "ReqModule" =>
        {"sys_bridge" => "sys_bridge.v",
        },

        "Connect" => { BRIDGE_a => [\&subst_infix,
["(.*)", ""],],
                    },
        "SysConn" => { BRIDGE_a => [\&getGlobSysInst, "E",
[\&subst_infix,  ["sys_(.*)", ""],],],
                    },
        "SysWrap" => { BRIDGE_a => [\&subst_infix,
["syswrap_(.*)", ""],],
                    },
        ####
        # Construction des noms des Blocs Systèmes dédiés
à un Composant et
        # vérification de la connectivité des interfaces
entre les Blocs Système et
        # leurs Composants dédiés
        "getOwnDest"  => \&getSpecSysOwn,
        "genVlogInstParameter" =>
\&gen_sys_VlogInstParameter,
        ####
      },

    }, #### BRIDGE_sys
  );
###
%SysSpecMap =
  (
  ######
  "FBUS_XCHG" =>
  ######
  { "MatchExpr"    => "^(?:.*_)(FBUS_XCHG)\$",
```

Variables faisant référence à un code générique commun pour l'ensemble des projets

4 de A2

```
    "ExtrExpr"    => "^(?:.*)([0-3])_FBUS_XCHG\$",
    "BaseIidExpr" => "^.*(FBUS_XCHG)\$",
   ###
    "IND_CON" =>
      { "ReqModule" =>
        {"fbus_xchg" => "fbus_xchg.v",
        },

        "Port" => { "FBUS" => ["FBUS_xchg",  "fbus_xchg",
0],
              "FBUSA" => ["FBUS_xchg",  "fbus_xchg", 1],
              },
        "Connect" => { "FBUS"   => [\&subst_infix,
["(XX.*)",    ""],],
                "FBUSA"  => [\&subst_infix,  ["in(.*)",
"aa_con"], ["out(.*)", "bb_con"],],
              },
        "SelfConnect"   => { "FBUS"   => [\&subst_infix,
["(XX.*)",    ""],],
                        "FBUSA"  => [\&subst_infix,
["out(.*)", "aa_con"], ["in(.*)", "bb_con"],],
              },
      "genOwn"      => \&genOwnSysSpecMapGeneric,
      "getOwnDest"  => \&getSpecSysOwn,

      "CfgCpp"  => [Fbus_Xchg, [ ['Own', [fbus_xchg,
IND_CON, ], ['None'], ],
                        ],
               ],
      },
    },
    ######
    'FBUS_PROBE' =>
    ######
    { "MatchExpr" => "^(.*)(FBUS_PROBE)\$",
      "ExtrExpr"  => "^.*?([0-3]).*FBUS_PROBE\$",
      "BaseIidExpr" => "^(.*)_FBUS_PROBE\$",
      ####
      "PROBE"   =>
      { "ReqModule" => {"fbus_probe" => "fbus_probe.v"
                },
       "Connect"   => {"FBUS"  => [\&subst_infix, ["(.*)",
""], ],
              },
       "Port" => { "FBUS"  => ["FBUS_probe",
"fbus_probe", 0],
              },
       "SysConn" => { "FBUS_probe" => \@std_glob_con,
```

```
                    },
        "genVlogInstParameter" =>
\&gen_sys_VlogInstParameter_Generic,
          },
      },
);
###
%indTypeCftpMap =                                    5 de A2
(
 "FBUS_xchg" => "FBUS_XCHG",
 "fbus_xchg" => "FBUS_XCHG",
 );

###
@sysGlobConnTab =          référencé par %SysPinConst
  (
   [ sys_RESET        ,  RESET     ],
   [ sys_POWER_GOOD   ,   POWER_GOOD ],
   [ sys_CLK          ,   CLK_33MHz     ],
   [ sys_CLK_2X       ,   CLK_66MHz     ],
  );
###
%SysPinConst =                                       6 de A2
    (
     "$GlobSysInst" =>  \@sysGlobConnTab,
     CPU_sys  => \@sysGlobConnTab,
     CMEM_sys  => \@sysGlobConnTab,
     CIO_sys   => \@sysGlobConnTab,
     BRIDGE_sys   => \@sysGlobConnTab,
     BRIDGE => \@sysGlobConnTab,
     );

#####
sub gen_sys_VlogInstParameter {                      7 de A2
  my($inst, $cat, $Cftp, $i_map) = @_;
  ($Cftp, $i_map) = getCftp($inst) unless $Cftp and
$i_map;
  my($extr) = $ {$i_map}{$Cftp}{'ExtrExpr'};
  $inst =~ $extr;
  my($Module_num, $Node_num, $SubNode_num) = ($1, $2, $3,
$4);
  my($parent_inst, $parent_cftp, $parent_map) =
&get_parent_inst($inst);
  my($num_add) = 0;
  $num_add = $ {$parent_map}{$parent_cftp}{'NumAdd'},
  $Node_num += $num_add
    if exists $ {$parent_map}{$parent_cftp}{'NumAdd'};
  my($parameter) = "";
```

```
    foreach $pp ($Module_num, $Node_num, $SubNode_num) {
      $parameter .= ", " if $parameter ne "" and $pp =~
/\d+/;
      $parameter .= $pp if $pp =~ /\d+/;
    }
    my($conn_mask) =
&get_parent_inst_connected_mask($inst);
    if($conn_mask ne ""){
        $parameter .= ", " if $parameter ne "";
        $parameter .= $conn_mask;
    }
    if($parameter){
      return "#($parameter)";
    }
    return ""
}
#######
1; ### makes perl happy
```

ANNEXE

A3 - Procédures de connexion des Composants

```
#! /usr/triton/bin/perl
##############################################################
#######################
#
#      Copyright (c) 2000 BULL - Worldwide Information
Systems
#                          All rights reserved
#
#  Module name   :  patent_connect_defs.pm
#  Author        :  Andrzej Wozniak
#
#  Description   : connection generation
#
#  Revision : $Revision: 1.1 $
#
#  Rev Date : $Date: 2001-01-29 20:19:10+01 $
#
##############################################################
#######################

####                                                  1 de A3
sub genDest {
  my($inst, $extExpr, $srcIfc) = @_;
  my(%dest) = ();
  my($dft);
  unless ( $inst =~ /$extExpr/ ){
   &dprint_config_error("genDest",
                "number extracting expression \"$extExpr\"
failed for $inst\n");
   return (\%dest, $dft);
  }
  while( $inst =~ /$extExpr/ ){
    $n1 = $1;
    $n2 = $2;
    $n3 = $3;
    my($nb) = 0;
    if($inst =~ $InstNameModuleMap{CPU}{"MatchExpr"}){
    ## CPU
      %dest = ("BRIDGE_$n1" => "FBUS"),
      $dft = "BRIDGE",
      last if $srcIfc =~ /^FBUS[A-Za-z0-9_]*/ ;
      goto label_error;
    }
    if($inst =~ $InstNameModuleMap{BRIDGE}{"MatchExpr"}){
    ## BRIDGE
```

```
            %dest = ( "CPU_$n1" => "FBUS"),
            $dft = "CPU",
            last if $srcIfc =~ /FBUS/;
            %dest = ( "CMEM_$n1"  => "CMEM"),
            $dft = "CMEM",
            last if $srcIfc =~ /CMEM/;
            %dest = ( "CIO_$n1"  => "CIO"),
            $dft = "CIO",
            last if $srcIfc =~ /CIO/;
            goto label_error;
          }
          if($inst =~ $InstNameModuleMap{CMEM}{"MatchExpr"}){
            ## CMEM
            %dest = ("BRIDGE_$n1" => "CMEM"),
            $dft = "BRIDGE",
            last if $srcIfc =~ /^CMEM$/;
            goto label_error;
          }
          if($inst =~ $InstNameModuleMap{CIO}{"MatchExpr"}){
            ## CIO
            %dest = ("BRIDGE_$n1" => "CIO"),
            $dft = "BRIDGE",
            last if $srcIfc =~ /^CIO$/;
            goto label_error;
          }
          &dprint_config_error("genDest", "unknown instance
        $inst");
          last;
        label_error:
          &dprint_config_error("genDest", "unknown interface
        $srcIfc for instance $inst");
          last;
        }
        return (\%dest, $dft);
        }
        #######
        1; ### makes perl happy
```

ANNEXE

A4 - Code de génération du fichier Verilog Top.v de connexion des Composants

```
#! /usr/triton/bin/perl
##############################################################
######################
#
#      Copyright (c) 2000 BULL - Worldwide Information
Systems
#                          All rights reserved
#
#  Module name   :  patent_verilog_defs.pm
#  Author        :  Andrzej Wozniak
#
#  Description   : VERILOG strings and defs
#                    for generation of top.v source code
#
#  Revision : $Revision: 1.4 $
#
#  Rev Date : $Date: 2001-02-19 19:55:41+01 $
#
##############################################################
######################

@verilog_src_path =
      (
       "$db_dir/patent/sim/models",
      );

$top_verilog_header = <<"END_OF_TOP_HEADER"
////////////////////////////////////////////////////////////
////////////
//  FILE \"%s\" GENERATED by A.W. PERL SCRIPT
//  FROM \"%s\"  file
////////////////////////////////////////////////////////////
////////////\n\n
//////
`timescale 100ps
END_OF_TOP_HEADER
;
;
$top_verilog_preamble = <<"END_OF_TOP_PREAMBLE"
//////
module top ();

wire        POWER_GOOD;
wire        RESET;
```

```
wire        CLK_33MHz;
wire        CLK_66MHz;

$GlobSysInstModule  $GlobSysInst(
        .sys_POWER_GOOD     (POWER_GOOD)
        .sys_RESET          (RESET),
        .sys_CLK            (CLK_33MHz),
        .sys_CLK_2X         (CLK_66MHz)
                    );
END_OF_TOP_PREAMBLE
;
####

$top_verilog_postamble = <<"END_OF_TOP_POSTAMBLE"

endmodule

//////////////
// END
//////////////
END_OF_TOP_POSTAMBLE
;

#######
1; ### makes perl happy
```

ANNEXE

A5 - Code de génération des Classes C++

```
#! /usr/triton/bin/perl
##############################################################
#######################
#
#       Copyright (c) 2000 BULL - Worldwide Information
Systems
#                            All rights reserved
#
#  Module name   : patent_cpp_gen_defs.pm
#  Author        :  Andrzej Wozniak
#
#  Description   : cpp generation tables
#
#  Revision : $Revision: 1.1 $
#
##############################################################
#######################

####
%moduleToCpPRClassMap =                                    1 de A5
  (
   fbus_probe => Probe_hwif,
  );
####
%classCppProtoMap =                                        2 de A5
  (
   ####
   CPU_Dut =>
   ####
   {
     Prea => "static CPU_Dut #SrcInst#
(LabelClass::#SrcIid#, TypeClass::#SrcIct#,\n"
             ."\t\t\t\tstring(\"#SrcVlogPath#\"));\n",
   },
   ####
   CPU_Monitor =>
   ####
   {
     Prea => "static CPU_Monitor #SrcInst#
(LabelClass::#SrcIid#",
     Iter => ",\n\t\t\t&#DstPort#",
     Post => ");\n",
     Idle => ",\n\t\t\t0",
   },
   ####
```

```
    Bridge_Dut =>
    ####
    {
      Prea => "static Bridge_Dut #SrcInst#
(LabelClass::#SrcIid#, TypeClass::#SrcIct#,\n"
             ."\t\t\t\tstring(\"#SrcVlogPath#\"));\n",
    },
    ####
    Cio_Dut =>
    ####
    {
      Prea => "static Cio_Dut #SrcInst#
(LabelClass::#SrcIid#, TypeClass::#SrcIct#,\n"
             ."\t\t\t\tstring(\"#SrcVlogPath#\"));\n",
    },
    ####
    Cmem_Dut =>
    ####
    {
      Prea => "static Cmem_Dut #SrcInst#
(LabelClass::#SrcIid#, TypeClass::#SrcIct#,\n"
             ."\t\t\t\tstring(\"#SrcVlogPath#\"));\n",
    },
    ####
    CPU_Xactor =>
    ####
    {
      Prea => "static CPU_Xactor #SrcInst#
(LabelClass::#SrcIid#",
      Iter => ",\n\t\t\t&#SrcPort#",
      Idle => ",\n\t\t\t0",
      Post => ");\n",
    },
    ####
    Bridge_Xactor =>
    ####
    {
      Prea => "static Bridge_Xactor #SrcInst#
(LabelClass::#SrcIid#",
      Iter => ",\n\t\t\t&#SrcPort#",
      Idle => ",\n\t\t\t0",
      Post => ");\n",
    },
    ####
    Fbus_hwif =>
    ####
    {
```

```
    Prea => "static Fbus_hwif #SrcInst#
    (LabelClass::#SrcIid#, TypeClass::#SrcIct#,\n"
                 ."\t\t\t\tstring(\"#SrcVlogPath#\"));\n",
       },
 5     ####
       Probe_hwif =>
       ####
       {
         Prea => "static Probe_hwif #SrcInst#
10   (LabelClass::#SrcIid#, TypeClass::PROBE,\n"
                 ."\t\t\t\tstring(\"#SrcVlogPath#\"));\n",
       },
       ####
       Cmem_hwif =>
15     ####
       {
         Prea => "static Cmem_hwif #SrcInst#
    (LabelClass::#SrcIid#, TypeClass::#SrcIct#,\n"
                 ."\t\t\t\tstring(\"#SrcVlogPath#\"));\n",
20     },
       ####
       Cio_hwif =>
       ####
       {
25       Prea => "static Cio_hwif #SrcInst#
    (LabelClass::#SrcIid#, TypeClass::#SrcIct#,\n"
                 ."\t\t\t\tstring(\"#SrcVlogPath#\"));\n",
       },
       ####
30     Fbus_Xchg =>
       ####
       {
         Prea => "static Fbus_Xchg #SrcInst#
    (LabelClass::#SrcIid#, TypeClass::#SrcIct#,\n"
35               ."\t\t\t\tstring(\"#SrcVlogPath#\"));\n",
       },
       );
    #####
    $cpp_preamble = <<"END_OF_PREAMBLE"
40  ////////////////////////////////////////
    //  FILE GENERATED by A.W. PERL SCRIPT
    //  FROM %s file
    //  FOR %s
    ////////////////////////////////////////\n\n
45  ////////////
    #include \"server_registry.hpp\"
    #include \"server_components.hpp\"
```

3 de A5

```
    //////////////\n\n
    const string ServerRegistry::m_serverName = \"%s\";
    const string ServerRegistry::m_configName = \"%s\";
    const int ServerRegistry::m_serverNumber = %d;
    Status InstantiateConfiguration()\{
    ///////////////\n
    END_OF_PREAMBLE
    ;
    ####
    $cpp_postamble = <<"END_OF_POSTAMBLE"
    \treturn Success;\n}
    //////////////
    // END
    //////////////
    END_OF_POSTAMBLE
    ;
    ####

    #######
    1; ### makes perl happy
```

4 de A5

ANNEXE

A6 – Fichier Résultat Verilog

```
////////////////////////////////////////////////////////////
//////////////
//  FILE "config_server_pat03_top.v" GENERATED by A.W.
PERL SCRIPT
//  FROM "patent/sim/configs/pat03.cfg"  file
////////////////////////////////////////////////////////////
//////////////

//////
`timescale 100ps
//////
module top ();

wire        POWER_GOOD;
wire        RESET;

wire        CLK_33MHz;
wire        CLK_66MHz;

Clock  SysClock(
        .sys_POWER_GOOD     (POWER_GOOD)
        .sys_RESET          (RESET),
        .sys_CLK            (CLK_33MHz),
        .sys_CLK_2X         (CLK_66MHz)
                    );
     ////////////////////////////////
     ///// Wire Declaration Section
     ////////////////////////////////
// wire        CLK_33MHz;          // output(1)
// wire        CLK_66MHz;          // input(3) output(1)
// wire        POWER_GOOD;         // input(3) output(1)
// wire        RESET;         // input(3) output(1)
 wire     [3:0]     W1_00_inXXack;           // input(1)
output(1)
 wire     [63:0]    W1_00_inXXadr_dat;       // input(1)
output(1)
 wire     [3:0]     W1_00_inXXreq;           // input(1)
output(1)
 wire     [3:0]     W1_00_outXXack;          // input(1)
output(1)
 wire     [63:0]    W1_00_outXXadr_dat;           //
input(1) output(1)
 wire     [3:0]     W1_00_outXXreq;          // input(1)
output(1)
 wire     [3:0]     W_00_XXack;         // inout(2)
```

```
    wire     [63:0]    W_00_XXadr_dat;          // inout(2)
    wire     [3:0]     W_00_XXreq;          // inout(2)
    wire     [31:0]    W_00_YYadr;          // input(1)
output(1)
    wire     [2:0]     W_00_YYctrl;         // input(1)
output(1)
    wire     [63:0]    W_00_YYdata;         // inout(2)
    wire     [1:0]     W_00_ZZack;          // input(1)
output(1)
    wire     [15:0]    W_00_ZZdata;         // inout(2)
    wire     [1:0]     W_00_ZZreq;          // input(1)
output(1)
        ////////////////////////////////
    wire          W_00_clk_2xn;       // input(1) output(1)
    wire          W_00_clk_2xp;       // input(1) output(1)
    wire          W_00_clkn;          // input(1) output(1)
    wire          W_00_clkp;          // input(1) output(1)
    wire     [3:0]     W_00_inXXack;        // input(1)
output(1)
    wire     [63:0]    W_00_inXXadr_dat;        // input(1)
output(1)
    wire     [3:0]     W_00_inXXreq;        // input(1)
output(1)
    wire     [3:0]     W_00_outXXack;           // input(1)
output(1)
    wire     [63:0]    W_00_outXXadr_dat;       // input(1)
output(1)
    wire     [3:0]     W_00_outXXreq;           // input(1)
output(1)
    wire          W_00_powergood;          // input(1)
output(1)
    wire          W_00_reset;         // input(1) output(1)
        ////////////////////////////////
        ///// Module Instancies Section
        ////////////////////////////////

//// BRIDGE_0_CPU_0_FBUS_XCHG -> IND_CON -> FBUS_xchg
////////////
fbus_xchg      BRIDGE_0_CPU_0_FBUS_XCHG (
            .XXadr_dat          (W_00_XXadr_dat),
            .XXreq        (W_00_XXreq),
            .XXack        (W_00_XXack),
            .inXXadr_dat        (W1_00_outXXadr_dat),
            .outXXadr_dat       (W1_00_inXXadr_dat),
            .inXXreq      (W1_00_outXXreq),
            .outXXreq           (W1_00_inXXreq),
            .inXXack      (W1_00_outXXack),
            .outXXack           (W1_00_inXXack));
```

```
//// CMEM_0 -> DUT -> CMEMD ////////////
cmem      CMEM_0 (
            .YYadr          (W_00_YYadr),
            .YYdata         (W_00_YYdata),
            .YYctrl         (W_00_YYctrl),
            .clk            (CLK_66MHz),
            .reset          (RESET),
            .powergood          (POWER_GOOD));

//// CIO_0 -> DUT -> CIOD ////////////
cio       CIO_0 (
            .ZZdata         (W_00_ZZdata),
            .ZZreq          (W_00_ZZreq),
            .ZZack          (W_00_ZZack),
            .clk            (CLK_66MHz),
            .reset          (RESET),
            .powergood          (POWER_GOOD));

//// BRIDGE_0 -> DUT_CORE -> BRD ////////////
bridge_core         BRIDGE_0 (
            .inXXadr_dat        (W1_00_inXXadr_dat),
            .outXXadr_dat       (W1_00_outXXadr_dat),
            .inXXreq        (W1_00_inXXreq),
            .outXXreq           (W1_00_outXXreq),
            .inXXack        (W1_00_inXXack),
            .outXXack           (W1_00_outXXack),
            .YYadr          (W_00_YYadr),
            .YYdata         (W_00_YYdata),
            .YYctrl         (W_00_YYctrl),
            .ZZdata         (W_00_ZZdata),
            .ZZreq          (W_00_ZZreq),
            .ZZack          (W_00_ZZack),
            .clk_2xp        (W_00_clk_2xp),
            .clk_2xn        (W_00_clk_2xn),
            .clkp           (W_00_clkp),
            .clkn           (W_00_clkn),
            .reset          (W_00_reset),
            .powergood          (W_00_powergood));

//// CPU_0_BRIDGE_0_FBUS_XCHG -> IND_CON -> FBUS_xchg
////////////
fbus_xchg       CPU_0_BRIDGE_0_FBUS_XCHG (
            .XXadr_dat          (W_00_XXadr_dat),
            .XXreq          (W_00_XXreq),
            .XXack          (W_00_XXack),
            .inXXadr_dat        (W_00_outXXadr_dat),
            .outXXadr_dat       (W_00_inXXadr_dat),
```

```
                .inXXreq        (W_00_outXXreq),
                .outXXreq           (W_00_inXXreq),
                .inXXack        (W_00_outXXack),
                .outXXack           (W_00_inXXack));
 5
    //// CPU_0 -> XACTOR -> FBUS_p ////////////
    fbus_p          CPU_0_XACTOR_FBUS_p (
                .inXXadr_dat        (W_00_inXXadr_dat),
                .outXXadr_dat       (W_00_outXXadr_dat),
10              .inXXreq        (W_00_inXXreq),
                .outXXreq           (W_00_outXXreq),
                .inXXack        (W_00_inXXack),
                .outXXack           (W_00_outXXack),
                .clk            (CLK_66MHz),
15              .reset          (RESET),
                .powergood          (POWER_GOOD));

    //// BRIDGE_0_sys -> SYS_CON -> BRIDGE_sys ////////////
    sys_bridge      #(0, 32'h00000007)  BRIDGE_0_sys (
20              .clk_2xp        (W_00_clk_2xp),
                .clk_2xn        (W_00_clk_2xn),
                .clkp           (W_00_clkp),
                .clkn           (W_00_clkn),
                .reset          (W_00_reset),
25              .powergood          (W_00_powergood),
                .sys_CLK_2X         (CLK_66MHz),
                .sys_CLK        (CLK_33MHz),
                .sys_RESET          (RESET),
                .sys_POWER_GOOD         (POWER_GOOD));
30
    endmodule

    ////////////
    // END
35  ////////////
```

ANNEXE

A7 - Fichiers résultat C++

```
////////////////////////////////////////
//  FILE GENERATED by A.W. PERL SCRIPT
//  FROM patent/sim/configs/pat03.cfg file
//  FOR server
////////////////////////////////////////


//////////////
#include "server_registry.hpp"
#include "server_components.hpp"


//////////////


const string ServerRegistry::m_serverName = "SERVER";
const string ServerRegistry::m_configName = "pat03";
const int ServerRegistry::m_serverNumber = 1;
Status InstantiateConfiguration(){
//////////////

static Fbus_hwif CPU_0_XACTOR_FBUS_p (LabelClass::CPU_0,
TypeClass::FBUS_type,
                    string("top.CPU_0_XACTOR_FBUS_p"));
static Cio_Dut CIO_0 (LabelClass::CIO_0, TypeClass::DUT,
                    string("top.CIO_0"));
static Cmem_Dut CMEM_0 (LabelClass::CMEM_0,
TypeClass::DUT,
                    string("top.CMEM_0"));
static Bridge_Dut BRIDGE_0 (LabelClass::BRIDGE_0,
TypeClass::DUT_CORE,
                    string("top.BRIDGE_0"));
static CPU_Xactor CPU_0_XACTOR (LabelClass::CPU_0,
TypeClass::XACTOR,
                    &CPU_0_XACTOR_FBUS_p);
static CPU_Monitor CPU_0_MONITOR (LabelClass::CPU_0,
TypeClass::MONITOR,
                    &CPU_0_XACTOR_FBUS_p);
static Fbus_Xchg BRIDGE_0_CPU_0_FBUS_XCHG
(LabelClass::BRIDGE_0_CPU_0, TypeClass::FBUS_XCHG,

     string("top.BRIDGE_0_CPU_0_FBUS_XCHG"));
static Fbus_Xchg CPU_0_BRIDGE_0_FBUS_XCHG
(LabelClass::CPU_0_BRIDGE_0, TypeClass::FBUS_XCHG,
```

```
        string("top.CPU_0_BRIDGE_0_FBUS_XCHG"));
        return Success;
    }
5   //////////////
    // END
    //////////////
```

REVENDICATIONS

1. Procédé d'établissement automatique, au moyen d'un système de traitement de données (40) associé à un programme dit Configurateur pour constituer un modèle global de simulation d'une architecture comprenant des modèles de circuits intégrés en développement pouvant constituer, à l'aide du Configurateur automatique, une machine ou une partie d'une machine et des modèles d'environnement de simulation, permettant de tester et de vérifier le circuit en développement, d'un fichier de définition de configuration (FCONF) de composants de l'architecture, ces composants constituant des blocs fonctionnels déterminés de description des fonctionnalités de circuits intégrés ou de parties de circuits intégrés, les composants étant choisis par l'utilisateur dans une bibliothèque de différents types de composants et une bibliothèque de composants d'environnements pour constituer le modèle global de l'architecture répondant à la spécification fonctionnelle définie dans le fichier de définition de configuration (FCONF) et conforme à la spécification de l'architecture du modèle global spécifié par un fichier de description de l'architecture (FDARCH), procédé caractérisé par le fait qu'il comporte les étapes suivantes:

- lecture du fichier de description d'architecture (FDARCH) du modèle global et mémorisation, dans une table de composants et de règles de connexion (TCRC), dans une table de règles de cohérence de connexions (TRCOH) et dans une table de formatage de fichiers source (TMFT), des informations relatives à l'ensemble des configurations possibles, chaque composant obtenant un nom (LABEL) identifiant, de façon non équivoque, sa position dans l'architecture, ainsi qu'un type parmi plusieurs types (Composants actifs, Blocs de Monitoring et de Vérification, Blocs intermédiaires, Blocs systèmes et Blocs Globaux),

- instanciation des composants, spécifiés dans le fichier de définition de configuration (FCONF) par l'utilisateur-concepteur au moyen d'une liste de composants présents désignés par leurs nom et type et

comportant des paramètres ou faisant appel à des procédures, le fichier de définition de la configuration (FCONF) comprenant un fichier de sélection des composants et de leur type et des indications supplémentaires optionnelles concernant le type d'interface et le serveur concerné par la configuration à générer par le Configurateur, et mémorisation des informations correspondantes dans une table de connexion des instances (TCINST),

- connexion topologique des instances et mémorisation des informations correspondantes dans une table de connexion des instances (TCINST),

- connexion physique des signaux d'interface, au niveau de chaque instance des composants, par application d'expressions régulières, mémorisées dans la table de composants et de règles de connexion (TCRC) portant sur le nom des signaux constituant une table de câblage (TCAB),

- utilisation de la table de connexion des instances (TCINST), de la table de câblage (TCAB) et de la table de formatage (TFMT) pour générer automatiquement des fichiers source de type HDL (MGHDL) et de type HLL (MGHLL) du modèle global de simulation, correspondant à la configuration spécifiée par le fichier de définition de configuration (FCONF).

2. Procédé selon la revendication 1, dans lequel le système Configurateur transmet aux parties de type HLL de chaque composant comprenant des informations sur :

- le nom (LABEL) du composant,
- le type de l'instance (DUT, XACTOR, VERIFIER, MONITOR),
- le chemin HDL, à savoir le nom hiérarchique du composant dans la description du modèle.

3. Procédé selon la revendication 1 ou 2, dans lequel le fichier de définition de la configuration (FCONF) comporte en plus un mot–clé (serveur<n>) indiquant le nom ou numéro du serveur sur lequel se trouve instancié un composant dans le cas d'une utilisation du procédé sur un système multi-serveur.

4. Procédé selon la revendication 3, dans lequel, dans le cas d'une utilisation multiserveur, le système configurateur exécute les étapes suivantes :

- découpage de la Configuration en plusieurs parties (de type HDL et de type HLL) en triant les composants de type HDL et les objets HLL selon leurs appartenances aux serveurs,
- génération des composants de type HDL périphériques servant à l'envoi et la réception des signaux entre les parties de la configuration,
- duplication des Blocs Globaux par le système Configurateur et instanciation des Blocs Globaux dupliqués sur chaque serveur,
- génération des parties de type HLL servant de support de communication entre les serveurs.

5. Procédé selon la revendication 3 ou 4, dans lequel la connexion automatique entre les composants par le système Configurateur comprend plusieurs phases :

- une phase topologique de haut niveau réalisant la sélection des composants et leurs positionnements respectifs,
- une phase de câblage réalisant la connexion effective entre les composants, cette phase générant comme résultat une table de câblage (TCAB) associant les signaux connectés ensemble, au nom unique du fil les connectant,
- une phase de génération des fichiers sources de type HDL et de type HLL.

6. Procédé selon la revendication 5, dans lequel la phase de câblage est effectuée par le système Configurateur selon les trois étapes suivantes :

a. les Blocs Globaux et les Blocs Système sont connectés en premier à tous les composants,

b. viennent ensuite les connexions des signaux entre les autres composants,

c. à la fin du câblage, une passe supplémentaire permet de connecter les signaux restant non connectés de chaque composant à des signaux prédéterminés pour assurer un état stable et déterminé, le système Configurateur générant alors des configurations partielles comprenant un sous-ensemble de l'architecture.

7. Procédé selon la revendication 6, dans lequel les signaux prédéterminés sont les signaux du Bloc Système correspondant au composant.

8. Procédé selon une des revendications 1 à 7, dans lequel le fichier de description de l'architecture (FDARCH) du modèle global comprend les modèles de simulation des Blocs globaux et des Blocs Système, ces deux types de composants étant connectés entre eux et gérant des signaux d'environnement.

9. Procédé selon la revendication 8, dans lequel les Blocs Système sont connectés aux autres composants et leur fournissent des signaux système qui leur sont spécifiques.

10. Procédé selon la revendication 9, dans lequel le système de traitement de données (40) effectue un contrôle de conformité des connexions en comparant la table de connexion des instances (TCINST) réelles entre blocs avec la table de règles de cohérence de connexions (TRCOH).

11. Procédé selon la revendication 10, dans lequel le système de traitement de données (40) compare les connexions physiques entre les composants à la table de règles de cohérence de connexions (TRCOH), pour détecter des incompatibilités entre extrémités de connexions entre les composants, et, en pareil cas, il spécifie, et ajoute, dans la table de

connexion des instances (TCINST), un composant adaptateur (Bloc Intermédiaire) (101) inséré dans la connexion considérée.

12. Procédé selon la revendication 11, dans lequel le fichier de définition de configuration (FCONF) comprend des informations, spécifiées par un attribut, concernant l'utilisation de composants adaptateurs (Blocs intermédiaires) avec les instances des Composants actifs, dont les connexions sont comparées à la table de connexion des instances (TCINST), pour détecter des incompatibilités entre extrémités de connexions entre les composants, et, en pareil cas, il spécifie, et ajoute, dans la table de connexion des instances (TCINST), un autre composant adaptateur (Bloc intermédiaire) (102) inséré dans la connexion considérée.

13. Procédé selon la revendication 12, dans lequel le système de traitement de données (40) sélectionne certaines des connexions entre composants de la table de règles de cohérence de connexions (TRCOH) et spécifie, et ajoute, dans la table de connexion des instances (TCINST), des connexions supplémentaires constituant des dérivations aboutissant à des modèles supplémentaires respectifs, représentant des outils de surveillance (sondes) des connexions.

14. Procédé selon l'une des revendications 1 à 13, dans lequel, dans la phase de génération des fichiers sources, le système Configurateur génère les fichiers source en langage HDL (MGHDL) et en langage HLL (MGHLL) en s'appuyant sur le contenu de la table de composants et de règles de connexion (TCRC), la table de règles de cohérence de connexions (TRCOH), la table de formatage des fichiers source (TMFT), la table de connexion des instances (TCINST) et la table de câblage (TCAB).

15. Procédé selon la revendication 14, dans lequel le système de traitement de données (40) effectue un traitement par le système Configurateur, pour chaque variante de configuration, pour obtenir plusieurs modèles de simulation correspondant à la même spécification fonctionnelle,

mais écrits en une description comportant différents mélanges des langages de niveaux différents (HDL, HLL).

16. Procédé selon l'une des revendications 1 à 15, dans lequel le système de traitement de données (40) établit la spécification fonctionnelle du modèle global de simulations dans un format informatique compatible avec un langage de programmation de haut niveau (HLL) et en format compatible avec un langage de description du matériel (HDL).

17. Procédé selon une des revendications 15 ou 16, dans lequel le fichier de définition de configuration (FCONF) comprend, pour chaque composant, au moins une partie en langage de type HDL, ladite partie en langage de type HDL fournissant une interface vers d'autres modèles.

18. Procédé selon la revendication 17, dans lequel les modèles comprenant une partie en langage de type HLL comprennent des adaptateurs d'interface.

19. Procédé selon la revendication 18, dans lequel le système Configurateur choisit chaque modèle d'adaptateurs d'interface en fonction de la table de règles de cohérence de connexions (TRCOH).

20. Procédé selon la revendication 19, dans lequel les connexions des signaux physiques sont spécifiés par des "Ports", chaque port étant une sélection arbitraire des signaux de l'interface de type HDL d'un composant à l'aide des expressions régulières portant sur les noms de ces signaux, et étant constitué de paires expression régulière / expression de substitution, ces expression étant appliquées successivement au nom de chaque signal de l'interface de type HDL, et, si la substitution finale est identique pour deux signaux, ces derniers sont connectés entre eux, la connexion étant mémorisée dans la table de câblage (TCAB).

21. Procédé selon la revendication 20, dans lequel chaque adaptateur d'interface étant partagé entre plusieurs modèles connectés sur le même port, un seul de ces modèles émet des signaux sur ledit port.

22. Système de traitement de données (40) pour l'établissement automatique d'un modèle global de simulation d'une configuration de blocs fonctionnels déterminés, mutuellement reliés par des connexions d'interfonctionnement, pour constituer le modèle global de simulation d'une architecture comprenant des modèles de circuit intégrés en développement pouvant constituer une machine répondant à la spécification fonctionnelle d'une configuration, système caractérisé par le fait que le système de traitement de données (40) utilise un programme Configurateur (PROGCONF) qui comprend des moyens de réaliser une simulation du câblage par l'application d'expressions régulières mémorisées, d'utiliser un fichier de définition de configuration (FCONF) en langage de haut niveau, une table de composants et de règles de connexion (TCRC) décrivant les propriétés (type, Interfaces de type HDL, ports, constructeurs d'objets de classes HLL, etc...) des composants logiciels de simulation du circuit, une table de règles de cohérence de connexions (TRCOH) en langage de haut niveau (HLL), des moyens d'instancier les éléments résultants du fichier de définition de configuration (FCONF), un générateur du code HLL qui combine les paramètres des composants avec les règles de connexion.

23. Système selon la revendication 22, caractérisé en ce qu'il existe au moins cinq types de composants : les Composants actifs, les Blocs de Monitoring et Vérification, les Blocs intermédiaires, les Blocs Système et les Blocs Globaux.

24. Système selon la revendication 23, caractérisé en ce que le système est agencé pour effectuer un contrôle de conformité des connexions en comparant la table de connexion des instances (TCINST) avec une table de règles de cohérence des connexions (TRCOH) physiques entre les modèles choisis des blocs constituant le modèle global.

25. Système selon la revendication 24, caractérisé en ce qu'il est agencé pour comparer la table de connexion des instances (TCINST) réelles entre blocs à la table de cohérence des connexions (TRCOH), pour détecter des incompatibilités entre extrémités de connexions entre blocs, et, en pareil cas, pour spécifier, et ajouter, dans la table de règles de cohérence des connexions (TRCOH), un bloc fonctionnel adaptateur (Bloc Intermédiaire) (101) inséré dans la connexion considérée.

26. Système selon une des revendications 22 à 25, caractérisé en ce que la table de composants et de règles de connexion (TCRC), comprenant les propriétés des composants, contient des paramètres globaux communs à tous les types de composants et se présente sous forme d'une table répartie suivant une ou plusieurs tables, associatives ou non, dont les entrées sont des noms désignant l'ensemble des modèles possibles pour un même composant.

27. Système selon la revendication 26, caractérisé en ce que les tables associatives peuvent contenir la description, soit sous forme de suites de paramètres, ou bien sous forme de références à des procédures qui génèrent les valeurs requises, les entrées de ces tables associatives étant des noms désignant l'ensemble des modèles possibles pour un même composant et formant une chaîne de caractères contenant des identificateurs spéciaux prédéterminés, substitués par les valeurs calculées par le système Configurateur.

28. Système selon la revendication 27, caractérisé en ce qu'au moins trois sélecteurs indiquent l'instance à prendre en compte, cette dernière étant transmise comme paramètre à un constructeur d'un objet HLL :

- un premier sélecteur indique l'instance ("item") courante,
- un deuxième sélecteur précise l'instance connectée à l'autre extrémité du port,

- un troisième sélecteur indique l'instance composée correspondant au Composant actif contenant le port d'observation.

29. Système selon l'une des revendications 22 à 28, caractérisé en ce que le système Configurateur utilise une ou plusieurs tables de règles de cohérence de connexions (TRCOH), qui représentent les règles d'interconnexion entre les composants et d'insertion des composants intermédiaires, une ou plusieurs tables de composants et de règles de connexions (TCRC), qui représentent les règles de connexion niveau système et les règles de génération de connexions entre les signaux, et une ou plusieurs tables de formatage de fichiers source (TFMT), qui représentent les règles de génération des instances des objets de type HLL.

30. Système selon l'une des revendications 22 à 29, caractérisé en ce que le système Configurateur utilise :

- une classe de base en HLL permettant d'identifier de façon univoque chaque objet instancié et d'interroger la configuration,
- des moyens de génération et d'instanciation automatique des Blocs Système,
- des moyens des tableaux associant les signaux connectés ensemble au nom unique du filles connectant,
- des moyens d'utiliser un tableau de formatage pour générer les fichiers sources en HDL et HLL.

31. Système selon l'une des revendications 22 à 30, caractérisé en ce que l'opérateur spécifie fonctionnellement, dans la plus large mesure possible, la configuration dans le langage de plus haut niveau et il complète la spécification fonctionnelle par les composants en langage de plus bas niveau.

32. Système selon une des revendications 22 à 31, caractérisé en ce que les entrées suivantes du hachage définissent le Type du composant (par

exemple DUT (modèle HDL), XACTOR (transactor), MONITOR, VERIFIER ou autres) et, à chaque Type de Composant, correspond un hachage composé à son tour des entrées suivantes :

- une première entrée *ReqModule* - contient le nom du module HDL du composant et le nom du fichier source correspondant,
- une deuxième entrée *Connect* - est la définition de la méthode de sélection des signaux faisant partie d'un Port, cette description étant composée d'une suite d'entrées indexées par le nom du Port ; à chaque nom de Port, le configurateur associe un tableau d'expressions régulières et un pointeur sur une procédure de connexion des signaux qui gère l'application de ces expressions aux noms des signaux de l'interface du composant.

33. Système selon la revendication 32, caractérisé en ce que la structure générique des Composants actifs comprend un Bloc contenant la description HDL et un Bloc en HLL, fournissant les chemins d'accès aux ressources HDL et, le cas échéant, une description du bloc en HLL, l'ensemble des signaux du bloc HDL constituant l'interface du Bloc englobant, formée, d'une part, de Ports, qui sont des sélections logiques arbitraires des signaux d'une interface et, d'autre part, d'adaptateurs d'interface qui sont les parties logicielles assurant, sur chaque Port, la communication bi-directionnelle entre les parties en HLL et celles en HDL, les adaptateurs d'interface étant choisis par le système Configurateur.

34. Système selon la revendication 33, caractérisé en ce que les Ports sont spécifiés sous forme d'expressions régulières, qui servent à la fois à sélectionner les sous-ensembles de signaux à connecter et à définir les règles de connexions.

35. Système selon une des revendications 22 à 34, caractérisé en ce que le système Configurateur génère des Composants dits de Transfert qui

sont insérés de chaque côté de la coupure sur les serveurs, ces composants étant simplement des fils pour les entrées et des registres pour les sorties.

ABREGE

Inventeurs : Andrzej WOZNIAK

Déposant : BULL S.A.

La présente invention concerne un procédé et un système d'établissement d'un modèle global de simulation d'une architecture. En particulier, l'invention concerne un procédé d'établissement automatique, au moyen d'un système de traitement de données (40) associé à un programme dit Configurateur pour constituer un modèle global de simulation d'une architecture comprenant des modèles de circuits intégrés en développement, procédé caractérisé par le fait qu'il comporte les étapes suivantes:

- lecture du fichier de description d'architecture (FDARCH) du modèle global et mémorisation des informations relatives à l'ensemble des configurations possibles,
- instanciation des composants et mémorisation des informations correspondantes dans une table de connexion des instances (TCINST),
- connexion topologique des signaux d'interface,
- génération automatique des fichiers source de type HDL (MGHDL) et de type HLL (MGHLL) du modèle global de simulation, correspondant à la configuration spécifiée par le fichier de définition de configuration (FCONF).

Figure 2A